

[illegible]


```
1 0001 0 MODULE DBGEVENT (IDENT = 'V04-000') =
2 0002 0
3 0003 1 BEGIN
4 0004 1
5 0005 1 *****
6 0006 1 *
7 0007 1 *   COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
8 0008 1 *   DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
9 0009 1 *   ALL RIGHTS RESERVED.
10 0010 1 *
11 0011 1 *   THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
12 0012 1 *   ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
13 0013 1 *   INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
14 0014 1 *   COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
15 0015 1 *   OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
16 0016 1 *   TRANSFERRED.
17 0017 1 *
18 0018 1 *   THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
19 0019 1 *   AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
20 0020 1 *   CORPORATION.
21 0021 1 *
22 0022 1 *   DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
23 0023 1 *   SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
24 0024 1 *
25 0025 1 *
26 0026 1 *****
27 0027 1
28 0028 1 WRITTEN BY
29 0029 1   Sidney R Maxwell III   May, 1982
30 0030 1
31 0031 1 MODULE FUNCTION
32 0032 1   This module contains the support routines for the new event-points.
33 0033 1   Included are the syntax and semantics routines for processing the
34 0034 1   new command forms, and the support for the new data structures.
35 0035 1
36 0036 1
37 0037 1 REQUIRE 'SRC$:DBGPROLOG.REQ';
38 0171 1
39 0172 1 LIBRARY 'LIB$:DBGGEN.L32';
40 0173 1
41 0174 1 FORWARD ROUTINE
42 0175 1   DBG$EVENT_INITIALIZATION : NOVALUE, | Initialize the event structures
43 0176 1   DBG$EVENT_SYNTAX, | Handle BREAK/TRACE/WATCH syntax
44 0177 1   DBG$EVENT_SEMANTICS, | Handle BREAK/TRACE/WATCH semantics
45 0178 1   DBG$EVENT_SHOW_CANCEL_SYNTAX, | [SHOW|CANCEL] BREAK/TRACE/WATCH syntax
46 0179 1   DBG$EVENT_SHOW_CANCEL_SEMANTICS, | Handle BREAK/TRACE/WATCH semantics
47 0180 1   DBG$EVENT_CANCEL_ALL : NOVALUE, | Cancel all eventpoints
48 0181 1   DBG$ACTIVATE_EVENTS : NOVALUE, | Activate all set events
49 0182 1   DBG$ACTIVATE_EVENT : NOVALUE, | Activate a set event
50 0183 1   DBG$DEACTIVATE_EVENTS : NOVALUE, | Deactivate all set events
51 0184 1   DBG$DEACTIVATE_EVENT : NOVALUE, | Deactivate a set event
52 0185 1   DBG$EXCEPTION_HANDLER, | Event Exception Handler
53 0186 1   DBG$PROCESS_EVENT, | Process an event
54 0187 1   DBG$UPDATE_WATCHPOINTS : NOVALUE, | Update the current value of each watchpoint
55 0188 1   ANNOUNCE_EVENT : NOVALUE, | Announce an event
56 0189 1   INSQUE1 : NOVALUE, | Insert a queue entry (1st vs 0th)
57 0190 1   REMQUE1 : NOVALUE, | Remove a queue entry (1st vs 0th)
```

DBGEVENT
V04-000

K 5
16-Sep-1984 00:59:10 VAX-11 Bliss-32 V4.0-742
14-Sep-1984 12:16:53 [DEBUG.SRC]DBGEVENT.B32;1

Page 2
(1)

:	58	0191	1	SHOW_EVENT_ENTRY: NOVALUE,	:	Show an Event Entry
:	59	0192	1	DELETE_EVENT_ENTRY: NOVALUE,	:	Delete an Event Entry (et al)
:	60	0193	1	PRINT_SOURCE: NOVALUE,	:	Print a source line
:	61	0194	1	PRINT_PC_AND_INSTRUCTION: NOVALUE,	:	Print PC and instruction
:	62	0195	1	PRINT_PC_INST_HANDLER,	:	Handler for above
:	63	0196	1	COMPARE_VMSDESC,	:	???
:	64	0197	1	GET_WATCH_POINT_VALUE: NOVALUE,	:	Get the value of the watch pointing
:	65	0198	1		:	variable
:	66	0199	1	PRIM_TO_VAL: NOVALUE,	:	Local cover routine for DBG\$PRIM_TO_VAL
:	67	0200	1	PRIM_TO_VAL_HANDLER,	:	Handler for above
:	68	0201	1	PARSE_WHEN_CONDITION: NOVALUE,	:	Handler to catch the bad status
:	69	0202	1		:	from parsing the WHEN clause
:	70	0203	1	MATCH_RIGHT_CALL_FRAME;	:	Make sure the call frame we got is
:	71	0204	1		:	the one we set bit 15 in saved
:	72	0205	1		:	PSW

74	0206	1	EXTERNAL ROUTINE	
75	0207	1	DBGSNGET_ADDRESS,	Get and address from a primary
76	0208	1	DBGSMAKE_VAL_DESC,	VMS desc to valdesc
77	0209	1	DBGS PRIM_TO_VAL,	Get a value descriptor from a primary
78	0210	1	DBGSNGET_SYMID,	Get a SYMID
79	0211	1	DBGSNCOPY_DESC,	Copy a primary descriptor
80	0212	1	DBGSNFREE_DESC,	Free a primary descriptor
81	0213	1	DBGSSTA_LOCK_SYMID,	Lock a SYMID
82	0214	1	DBGSSTA_UNLOCK_SYMID,	Unlock a SYMID
83	0215	1	DBGS PRINT,	'Print' an FAO string
84	0216	1	DBGSNEWLINE,	Dump 'printed' buffer
85	0217	1	DBGS PRINT_VALUE,	'Print' a value from a val desc
86	0218	1	DBGS PRINT_IDENTIFIER,	'Print' an identifier from a primary
87	0219	1	DBGS PRINT_IDENTIFIER_PC,	'Print' an identifier from an address
88	0220	1	DBGS PUTMSG,	Put an exception message
89	0221	1	DBGS PC TO LINE LOOKUP,	Match a PC to line
90	0222	1	DBGS SRC_TYPE_PC_SOURCE: NOVALUE,	Display source line
91	0223	1	DBGS INS_DECODE,	Decode [and display] instruction
92	0224	1	DBGS OPCODE_INDEX,	Get opcode from mnemonic
93	0225	1	DBGS FINAL_HANDLER,	DEBUG's final handler
94	0226	1	DBGS READ_ACCESS,	Check for read access
95	0227	1	DBGS MAP_TO_REG_ADDR,	Map an address to register space
96	0228	1	DBGSNGET_PAGES,	Get pages from a primary
97	0229	1	DBGS DATA_LENGTH,	Get data bit length from VMS descriptor
98	0230	1	DBGS IS_IT_ENTRY,	Checks address for entry point
99	0231	1	DBGSNMAKE_ARG_VECT,	Constructs a message argument vector
100	0232	1	DBGSNSAVE_DECIMAL_INTEGER,	Saves an inline numeric literal
101	0233	1	DBGSNSAVE_BREAK_BUFFER: NOVALUE,	Saves away a break action buffer
102	0234	1	DBGSGET_MEMORY,	Get a block of permanent memory
103	0235	1	DBGSREL_MEMORY,	Release a block of permanent memory
104	0236	1	DBGSGET_TEMPMEM,	Get a block of temporary memory
105	0237	1	DBGSREL_TEMPMEM,	Release all of temporary memory
106	0238	1	DBGS RST_TEMP RELEASE: NOVALUE,	Release all temporary RST entries
107	0239	1	DBGSNNEXT_WORD,	Isolates next word for syntax errors
108	0240	1	DBGSNMATCH,	Routine to match keywords
109	0241	1	DBGSNPARSE_ADDRESS,	Interface routine to Address Expression Interpreter
110	0242	1	DBGSNPARSE_EXPRESSION,	Parse an expression
111	0243	1	DBGSNTYPE_CONV,	Type converter
112	0244	1	DBGSWRITE_MEM,	Write into memory
113	0245	1	DBGSNCIS_ADD,	Add command string
114	0246	1	DBGS COMMAND PROC,	Execute a command list
115	0247	1	DBGSNINITIALIZE,	Initialize for user commands
116	0248	1	DBGSNSYNTAX_ERROR,	Constructs a syntax error message
117	0249	1	DBGSSET_STP_LVL: NOVALUE,	Set step level
118	0250	1	DBGS PSEUDO PROG,	DEBUG's call code
119	0251	1	LIB\$SIGNAL: ADDRESSING_MODE (GENERAL),	
120	0252	1	LIB\$STOP;	
121	0253	1		
122	0254	1	EXTERNAL	
123	0255	1	DBGSGB_CALL_NORMAL_RET: BYTE,	CALL command status flag; 0 = not in a CALL command, 1 = in a CALL that has not yet returned, 2 = in normal return from a CALL cmd
124	0256	1		
125	0257	1		
126	0258	1		
127	0259	1	DBGSGB_RADIX: VECTOR[3, BYTE],	Radix settings
128	0260	1	DBGSGB_LANGUAGE: BYTE,	Current set lang
129	0261	1	DBGSGL_SIGN_FLAG,	Flag set to print '+' before signed variable
130	0262	1		

131	0263	1	DBG\$GV_CONTROL: DBG\$CONTROL_FLAGS,	Major State of DEBUG
132	0264	1	DBG\$GB_STP_PTR:	Current stepping modes
133	0265	1	REF EVENTS\$STEPPING_DESCRIPTOR,	
134	0266	1	DBG\$GB_MOD_PTR: REF VECTOR[,BYTE],	Current mode flags
135	0267	1	DBG\$GL_CISREAD,	Head of command input stream
136	0268	1	DBG\$GL_CIS_LEVELS,	Number of levels of CIS nesting
137	0269	1	DBG\$GL_LIB_SIGNAL_ADDR,	Address of LIB\$SIGNAL
138	0270	1	DBG\$GL_LIB_STOP_ADDR,	Address of LIB\$STOP
139	0271	1	DBG\$GL_STEP_NUM,	Step count
140	0272	1	DBG\$GB_TAKE_CMD: BYTE,	User commands okay (no go etc.)
141	0273	1	DBG\$RUNFRAME: BLOCK[,BYTE],	Run frame
142	0274	1	DBG\$GB_EXC_BRE_FLAG: BYTE,	Encountered an exception break
143	0275	1	DBG\$GB_GO_ARG_FLAG: BYTE,	GO command had an argument
144	0276	1	DBG\$GB_NO_GLOBALS: BYTE,	Don't convert to global symbol
145	0277	1	DBG\$OPCODE_KIND_TABLE:	???
146	0278	1	VECTOR [512,WORD],	
147	0279	1	DBG\$OPCODE_NAME_TABLE:	???
148	0280	1	BLOCKVECTOR [,10,BYTE],	
149	0281	1	DBG\$PSEUDO_EXIT,	Label in pseudo code for CALL
150	0282	1	DBG\$PSEUDO_SSI,	Label in pseudo code for
151	0283	1		DBG_SSI_ROUTINE
152	0284	1	DBG\$GB_SET_SSI_CNT: BYTE,	Is watch pointing active?
153	0285	1	DBG\$TERM_HANDLER,	Label in termination handler
154	0286	1	DBG\$USER_EXIT,	Label in exit code
155	0287	1	PRIM_HANDLER_2,	Label in primary handler
156	0288	1	DBG\$GL_OUTPRAB: BLOCK[,BYTE],	RAB for 'OUTPUT'
157	0289	1	DBG\$GL_ORIG_COMMAND_PTR,	Pointer to original command string
158	0290	1	DBG\$GL_UPCASE_COMMAND_PTR: VECTOR[2],	
159	0291	1		Pointers to start and end
160	0292	1		of current command string
161	0293	1		
162	0294	1	GLOBAL	
163	0295	1	DBG\$GB_SET_BREAK_FLAG:	Set to TRUE during processing
164	0296	1	BYTE INITIAL(0),	of address in SET BREAK
165	0297	1	DBG\$GB_SET_WATCH_FLAG:	Set to TRUE during processing
166	0298	1	BYTE INITIAL(0);	of System service cycle
167	0299	1		(CALL to RET) for watch point
168	0300	1		value comparison
169	0301	1		
170	0302	1	OWN	
171	0303	1	INVALID_FLAG,	Set to TRUE when a Primary becomes
172	0304	1		invalid (PRIM_TO_VAL_HANDLER)
173	0305	1	NEWVALUE: REF DBG\$VALDESC,	New value for watch-point variable
174	0306	1	OLDVALUE: REF DBG\$VALDESC,	Old value for watch point variable
175	0307	1	TYPE_SOURCE,	It's okay to type source
176	0308	1	ACCVIO_PC,	PC at an ACCVIO
177	0309	1	WATCH_PC,	PC at next instruction after System
178	0310	1		service call when watch pointing
179	0311	1		is triggered, used to report watch
180	0312	1		point address
181	0313	1	SKIP_ACCVIO : INITIAL (FALSE),	We're [not initially] skipping ACCVIO's
182	0314	1	SKIP_WATCHES : INITIAL (FALSE),	Flag set to indicate We t-bit over RET
183	0315	1		of Service call, report watched
184	0316	1		value if there is one
185	0317	1	WHEN_CONDITION: REF DBG\$VALDESC;	Condition for WHEN clause
186	0318	1		
187	0319	1	BUILTIN	

DBGEVENT
V04-000

:	188	0320	1	INSQUE,
:	189	0321	1	REMQUE,
:	190	0322	1	PROBER;

N 5
16-Sep-1984 00:59:10 VAX-11 Bliss-32 V4.0-742
14-Sep-1984 12:16:53 [DEBUG.SRC]DBGEVENT.B32;1

Page 5
(2)

! Insert an entry on a list
! Remove an entry from a list
! Probe a location for read-access

```
: 192      0323 1 ! Define the QUEUE_HEAD data structure, used to reference the head of
: 193      0324 1 ! a queue.
: 194      0325 1
: 195      0326 1 FIELD
: 196      0327 1     QUEUE_HEAD_FIELDS =
: 197      0328 1     SET
: 198      0329 1         A_QUEUE_ENTRY = [0,A_],      ! Queue entry's address
: 199      0330 1
: 200      0331 1         L_QUEUE_FLINK = [0,L_],      ! Queue's 0th (normal) Forward LINK
: 201      0332 1         L_QUEUE_BLINK = [1,L_],      ! Queue's 0th (normal) Backward Link
: 202      0333 1
: 203      0334 1         L_QUEUE_FLINK1 = [2,L_],     ! Queue's 1st Forward LINK
: 204      0335 1         L_QUEUE_BLINK1 = [3,L_],     ! Queue's 1st Backward Link
: 205      0336 1     TES;
: 206      0337 1
: 207      0338 1
: 208      0339 1 MACRO QUEUE_HEAD = BLOCK [4, LONG] FIELD (QUEUE_HEAD_FIELDS)%;
: 209      0340 1
: 210      0341 1 GLOBAL
: 211      0342 1     EVENT$CMD_QUEUE:      QUEUE_HEAD,      ! Command Queue
: 212      0343 1     EVENT$DO_LIST_QUEUE:  QUEUE_HEAD,      ! DO List Queue
: 213      0344 1     EVENT$WHEN_QUEUE:     QUEUE_HEAD,      ! WHEN Queue
: 214      0345 1     EVENT$PAGE_QUEUE:     QUEUE_HEAD,      ! PAGE Queue
: 215      0346 1
: 216      0347 1
: 217      0348 1     ! The following is used to flag a stop, such as from a BREAK or the
: 218      0349 1     ! end of a STEP, as opposed to a TRACE....
: 219      0350 1     !
: 220      0351 1     EVENT$B_STOP_USER :      BYTE;
```



```

: 222      0352 1 OWN
: 223      0353 1 ! Opcode Table for /INSTRUCTION = (<mnemonic> ,...)
: 224      0354 1 !
: 225      0355 1 DBG$OPCODES_USER : BITVECTOR[512],
: 226      0356 1
: 227      0357 1
: 228      0358 1 ! Opcode Table for STEP/INSTRUCTION = (<mnemonic> ,...)
: 229      0359 1 !
: 230      0360 1 DBG$OPCODES_STEP_USER : BITVECTOR[512],
: 231      0361 1
: 232      0362 1
: 233      0363 1 ! Opcode Table for /CALL events.
: 234      0364 1 !
: 235      0365 1 DBG$OPCODES_CALL : BITVECTOR[512]
: 236      0366 1 PRESET
: 237      0367 1 ([XX'0FB'] = 1, ! CALLS
: 238      0368 1 [XX'0FA'] = 1, ! CALLG
: 239      0369 1 [XX'030'] = 1, ! BSBW
: 240      0370 1 [XX'010'] = 1, ! BSBB
: 241      0371 1 [XX'016'] = 1, ! JSB
: 242      0372 1 [XX'005'] = 1, ! RSB
: 243      0373 1 [XX'004'] = 1, ! RET
: 244      0374 1 ),
: 245      0375 1
: 246      0376 1
: 247      0377 1 ! Opcode Table for /BRANCH events.
: 248      0378 1 !
: 249      0379 1 DBG$OPCODES_BRANCH : BITVECTOR[512]
: 250      0380 1 PRESET
: 251      0381 1 ([XX'012'] = 1, ! BNEQ or BNEQU
: 252      0382 1 [XX'013'] = 1, ! BEQL or BEQLU
: 253      0383 1 [XX'014'] = 1, ! BGTR
: 254      0384 1 [XX'015'] = 1, ! BLEQ
: 255      0385 1 [XX'018'] = 1, ! BGEQ
: 256      0386 1 [XX'019'] = 1, ! BLSS
: 257      0387 1 [XX'01A'] = 1, ! BGTRU
: 258      0388 1 [XX'01B'] = 1, ! BLEQU
: 259      0389 1 [XX'01C'] = 1, ! BVC
: 260      0390 1 [XX'01D'] = 1, ! BVS
: 261      0391 1 [XX'01E'] = 1, ! BGEQU or BCC
: 262      0392 1 [XX'01F'] = 1, ! BLSSU or BCS
: 263      0393 1 [XX'011'] = 1, ! BRB
: 264      0394 1 [XX'031'] = 1, ! BRW
: 265      0395 1 [XX'017'] = 1, ! JMP
: 266      0396 1 [XX'0E0'] = 1, ! BBS
: 267      0397 1 [XX'0E1'] = 1, ! BBC
: 268      0398 1 [XX'0E2'] = 1, ! BBSS
: 269      0399 1 [XX'0E3'] = 1, ! BBBS
: 270      0400 1 [XX'0E4'] = 1, ! BBSC
: 271      0401 1 [XX'0E5'] = 1, ! BBCC
: 272      0402 1 [XX'0E6'] = 1, ! BBSSI
: 273      0403 1 [XX'0E7'] = 1, ! BBCCI
: 274      0404 1 [XX'0E8'] = 1, ! BLBS
: 275      0405 1 [XX'0E9'] = 1, ! BLBC
: 276      0406 1 [XX'09D'] = 1, ! ACBB
: 277      0407 1 [XX'03D'] = 1, ! ACBW
: 278      0408 1 [XX'0F1'] = 1, ! ACBL
```



```

: 279      0409 1      [%X'04F'] = 1, : ACBF
: 280      0410 1      [%X'06F'] = 1, : ACBD
: 281      0411 1      [%X'14F'] = 1, : ACBG
: 282      0412 1      [%X'16F'] = 1, : ACBH
: 283      0413 1      [%X'0F3'] = 1, : AOBLEQ
: 284      0414 1      [%X'0F2'] = 1, : AOBLSS
: 285      0415 1      [%X'0F4'] = 1, : SOBGEO
: 286      0416 1      [%X'0F5'] = 1, : SOBGTR
: 287      0417 1      [%X'08F'] = 1, : CASEB
: 288      0418 1      [%X'0AF'] = 1, : CASEW
: 289      0419 1      [%X'OCF'] = 1, : CASEL
: 290      0420 1      ),
: 291      0421 1
: 292      0422 1
: 293      0423 1      ! Opcode Table for stepping /OVER.
: 294      0424 1      !
: 295      0425 1      DBG$OPCODES_STEP_OVER : BITVECTOR[512]
: 296      0426 1      PRESET
: 297      0427 1      ([%X'0FB'] = 1, : CALLS
: 298      0428 1      [%X'0FA'] = 1, : CALLG
: 299      0429 1      [%X'030'] = 1, : BSBW
: 300      0430 1      [%X'010'] = 1, : BSBB
: 301      0431 1      [%X'016'] = 1, : JSB
: 302      0432 1      );

```



```

304      0433 1 MACRO
305      0434 1
306      0435 1 ! Calculate word length from bit length.
307      0436 1
308      M 0437 1 BIT_TO_WORD_LENGTH (BIT_LENGTH) =
309      M 0438 1 -((BIT_LENGTH + %BPVAL - 1) / %BPVAL)
310      0439 1 %,
311      0440 1
312      0441 1 ! Setup a UPLIT %ASCIC string.
313      0442 1
314      0443 1
315      M 0444 1 $AC (STRING) =
316      M 0445 1 UPLIT BYTE (%ASCIC STRING)
317      0446 1 %,
318      0447 1
319      0448 1
320      0449 1 ! Test <string> for a match with the input string.
321      0450 1
322      M 0451 1 MATCH (STRING) =
323      M 0452 1 (DBG$NMATCH(.INPUT_DESC, STRING,
324      0453 1 %IF %NULL (%REMAINING) %THEN 1 %ELSE %REMAINING %FI)) %,
325      0454 1
326      0455 1
327      0456 1 ! Match <string> to the input string, and give a syntax
328      0457 1 error if <string> does not match.
329      0458 1
330      M 0459 1 NEED_MATCH (STRING) =
331      0460 1 (IF NOT MATCH (STRING) THEN SYNTAX_ERROR;) %,
332      0461 1
333      0462 1
334      0463 1 ! Syntax Error
335      0464 1
336      M 0465 1 SYNTAX_ERROR =
337      M 0466 1 BEGIN
338      M 0467 1 .MESSAGE_VECT =
339      M 0468 1 (IF MATCH (DBG$CS_CR)
340      M 0469 1 THEN
341      M 0470 1 DBG$NMAKE_ARG_VECT (DBG$_NEEDMORE)
342      M 0471 1 ELSE
343      M 0472 1 DBG$NSYNTAX_ERROR (DBG$NNEXT_WORD (.INPUT_DESC))
344      M 0473 1 );
345      M 0474 1 RETURN ST$K_SEVERE
346      M 0475 1 END
347      0476 1 %,
348      0477 1
349      0478 1
350      0479 1 ! Allocate an adverb node.
351      0480 1
352      M 0481 1 GET_ADVERB_NODE =
353      M 0482 1 BEGIN
354      M 0483 1 ADVERB_NODE = DBG$GET_TEMPMEM (DBG$K_ADVERB_NODE_SIZE);
355      M 0484 1 ADVERB_NODE [DBG$L_ADVERB_LINK] = 0;
356      M 0485 1 .ADVERB_LINK = .ADVERB_NODE;
357      M 0486 1 ADVERB_LINK = ADVERB_NODE [DBG$L_ADVERB_LINK]
358      M 0487 1 END
359      0488 1 %,
360      0489 1
```

```

: 361      0490 1
: 362      0491 1
: 363      0492 1
: 364      M 0493 1
: 365      M 0494 1
: 366      M 0495 1
: 367      M 0496 1
: 368      M 0497 1
: 369      M 0498 1
: 370      M 0499 1
: 371      0500 1
: 372      0501 1
: 373      0502 1
: 374      0503 1
: 375      0504 1
: 376      M 0505 1
: 377      M 0506 1
: 378      M 0507 1
: 379      M 0508 1
: 380      M 0509 1
: 381      M 0510 1
: 382      M 0511 1
: 383      0512 1

```

```

! Allocate a noun node.
!
GET_NOUN_NODE =
  BEGIN
    NOUN_NODE = DBG$GET_TEMP_MEM (DBG$K_NOUN_NODE_SIZE);
    NOUN_NODE [DBG$L_NOUN_LINK] = 0;
    .NOUN_LINK = .NOUN_NODE;
    NOUN_LINK = NOUN_NODE [DBG$L_NOUN_LINK]
  END
%,

! Write an opcode to memory.
!
WRITE_OPCODE (ADDRESS, OPCODE) =
  (
    LOCAL
      OP;

    OP = OPCODE;
    DBG$WRITE_MEM (ADDRESS, OP, 1)
  )
%,

```



```

: 385      0513 1 ! Literals
: 386      0514 1 !
: 387      0515 1 LITERAL
: 388      0516 1
: 389      0517 1
: 390      0518 1
: 391      0519 1 ! Define the adverb literals, used to describe the contents
: 392      0520 1 ! of an adverb node.
: 393      P 0521 1 ENUMERATE (0, ADVERB_AFTER,
: 394      P 0522 1 ADVERB_TEMPORARY,
: 395      P 0523 1 ADVERB_SILENT,
: 396      P 0524 1 ADVERB_NOSILENT,
: 397      P 0525 1 ADVERB_WHEN,
: 398      P 0526 1 ADVERB_DO,
: 399      P 0527 1
: 400      P 0528 1 ADVERB_READ,
: 401      P 0529 1 ADVERB_WRITE,
: 402      P 0530 1 ADVERB_MODIFY,
: 403      P 0531 1 ADVERB_EXECUTE,
: 404      P 0532 1 ADVERB_RETURN,
: 405      P 0533 1 ADVERB_EXCEPTION,
: 406      P 0534 1 ADVERB_SOURCE,
: 407      P 0535 1 ADVERB_NOSOURCE,
: 408      P 0536 1 ADVERB_SYSTEM,
: 409      P 0537 1 ADVERB_NOSYSTEM,
: 410      P 0538 1 ADVERB_INT0,
: 411      P 0539 1 ADVERB_OVER,
: 412      P 0540 1 ADVERB_CALL,
: 413      P 0541 1 ADVERB_BRANCH,
: 414      P 0542 1 ADVERB_LINE,
: 415      P 0543 1 ADVERB_INSTRUCTION,
: 416      P 0544 1 ADVERB_OPCODE,
: 417      P 0545 1 ADVERB_ALL
: 418      0546 1 );
: 419      0547 1
```

```

421 0548 1 GLOBAL ROUTINE DBG$EVENT_INITIALIZATION: NOVALUE =
422 0549 1
423 0550 1 FUNCTION
424 0551 1     This routine is called to initialize the event structures when
425 0552 1     DEBUG is first entered.
426 0553 1
427 0554 1 INPUTS
428 0555 1     None
429 0556 1
430 0557 1 OUTPUTS
431 0558 1     None
432 0559 1
433 0560 1
434 0561 2 BEGIN
435 0562 2
436 0563 2
437 0564 2     ! Intialize the head of the Command Queue.
438 0565 2
439 0566 2     EVENT$CMD_QUEUE [L_QUEUE_FLINK] = EVENT$CMD_QUEUE;
440 0567 2     EVENT$CMD_QUEUE [L_QUEUE_BLINK] = EVENT$CMD_QUEUE;
441 0568 2     EVENT$CMD_QUEUE [L_QUEUE_FLINK1] = EVENT$CMD_QUEUE;
442 0569 2     EVENT$CMD_QUEUE [L_QUEUE_BLINK1] = EVENT$CMD_QUEUE;
443 0570 2
444 0571 2
445 0572 2     ! Initialize the head of the DO List Queue.
446 0573 2
447 0574 2     EVENT$DO_LIST_QUEUE [L_QUEUE_FLINK] = EVENT$DO_LIST_QUEUE;
448 0575 2     EVENT$DO_LIST_QUEUE [L_QUEUE_BLINK] = EVENT$DO_LIST_QUEUE;
449 0576 2
450 0577 2
451 0578 2     ! Initialize the head of the WHEN Queue.
452 0579 2
453 0580 2     EVENT$WHEN_QUEUE [L_QUEUE_FLINK] = EVENT$WHEN_QUEUE;
454 0581 2     EVENT$WHEN_QUEUE [L_QUEUE_BLINK] = EVENT$WHEN_QUEUE;
455 0582 2
456 0583 2
457 0584 2     ! Initialize the head of the PAGE Queue.
458 0585 2
459 0586 2     EVENT$PAGE_QUEUE [L_QUEUE_FLINK] = EVENT$PAGE_QUEUE;
460 0587 2     EVENT$PAGE_QUEUE [L_QUEUE_BLINK] = EVENT$PAGE_QUEUE;
461 0588 1 END;
```

```

.TITLE DBGEVENT
.IDENT \V04-000\
.PSECT DBG$OWN,NOEXE, PIC,2
```

```

00000 INVALID_FLAG:
00004 NEWVALUE:
00008 OLDVALUE:
0000C TYPE_SOURCE:
00010 ACCVIO_PC:
```



```
                                .BLKB 4
00014 WATCH_PC:
                                .BLKB 4
00000000 00018 SKIP_ACCVIOS:
                                .LONG 0
00000000 0001C SKIP_WATCHES:
                                .LONG 0
00020 WHEN_CONDITION:
                                .BLKB 4
00024 DBG$OPCODES_USER:
                                .BLKB 64
00064 DBG$OPCODES_STEP_USER:
                                .BLKB 64
30 000A4 DBG$OPCODES_CALL:
                                .BYTE 48
00 000A5                                .BYTE 0
41 000A6                                .BYTE 65
00# 000A7                                .BYTE 0[3]
01 000AA                                .BYTE 1
00# 000AB                                .BYTE 0[24]
0C 000C3                                .BYTE 12
00# 000C4                                .BLKB 32
00# 000E4 DBG$OPCODES_BRANCH:
                                .BYTE 0[2]
FF BE 000E6                                .BYTE -66, -1
00# 000E8                                .BYTE 0[2]
20 02 000EA                                .BYTE 2, 32
00 000EC                                .BYTE 0
80 000ED                                .BYTE -128
00# 000EE                                .BYTE 0[3]
80 000F1                                .BYTE -128
00# 000F2                                .BYTE 0[3]
80 000F5                                .BYTE -128
00 000F6                                .BYTE 0
20 000F7                                .BYTE 32
00 000F8                                .BYTE 0
80 000F9                                .BYTE -128
00# 000FA                                .BYTE 0[3]
80 000FD                                .BYTE -128
00# 000FE                                .BYTE 0[2]
3E 03 FF 00100                                .BYTE -1, 3, 62
00# 00103                                .BYTE 0[10]
80 0010D                                .BYTE -128
00# 0010E                                .BYTE 0[3]
80 00111                                .BYTE -128
00# 00112                                .BLKB 18
00# 00124 DBG$OPCODES_STEP_OVER:
                                .BYTE 0[2]
41 00126                                .BYTE 65
00# 00127                                .BYTE 0[3]
01 0012A                                .BYTE 1
00# 0012B                                .BYTE 0[24]
0C 00143                                .BYTE 12
00144                                .BLKB 32
```

```
.PSECT DBG$GLOBAL,NOEXE, PIC,2
```

```
00 00000 DBG$GB_SET BREAK FLAG::
      .BYTE 0
00 00001 DBG$GB_SET WATCH FLAG::
      .BYTE 0
00002      .BLKB 2
00004 EVENT$CMD_QUEUE::
      .BLKB 16
00014 EVENT$DO_LIST_QUEUE::
      .BLKB 16
00024 EVENT$WHEN_QUEUE::
      .BLKB 16
00034 EVENT$PAGE_QUEUE::
      .BLKB 16
00044 EVENT$B_STOP_USER::
      .BLKB 1
```

```
.EXTRN DBG$NGET_ADDRESS
.EXTRN DBG$MAKE_VAL_DESC
.EXTRN DBG$PRIM_TO_VAL
.EXTRN DBG$NGET_SYMID, DBG$NCOPY_DESC
.EXTRN DBG$NFREE_DESC, DBG$STA_LOCK_SYMID
.EXTRN DBG$STA_UNLOCK_SYMID
.EXTRN DBG$PRINT, DBG$NEWLINE
.EXTRN DBG$PRINT_VALUE
.EXTRN DBG$PRINT_IDENTIFIER
.EXTRN DBG$PRINT_IDENTIFIER_PC
.EXTRN DBG$PUTMSG, DBG$PC_TO_LINE_LOOKUP
.EXTRN DBG$SRC_TYPE_PC_SOURCE
.EXTRN DBG$INS_DECODE, DBG$OPCODE_INDEX
.EXTRN DBG$FINAL_HANDL
.EXTRN DBG$READ_ACCESS
.EXTRN DBG$MAP_TO_REG_ADDR
.EXTRN DBG$NGET_PAGES, DBG$DATA_LENGTH
.EXTRN DBG$IS_IT_ENTRY
.EXTRN DBG$NMAKE_ARG_VECT
.EXTRN DBG$NSAVE_DECIMAL_INTEGER
.EXTRN DBG$NSAVE_BREAK_BUFFER
.EXTRN DBG$GET_MEMORY, DBG$REL_MEMORY
.EXTRN DBG$GET_TEMP_MEM
.EXTRN DBG$REL_TEMP_MEM
.EXTRN DBG$RST_TEMP_RELEASE
.EXTRN DBG$NNEXT_WORD, DBG$NMATCH
.EXTRN DBG$NPARSE_ADDRESS
.EXTRN DBG$NPARSE_EXPRESSION
.EXTRN DBG$NTYPE_CONV, DBG$WRITE_MEM
.EXTRN DBG$NCIS_ADD, DBG$COMMAND_PROC
.EXTRN DBG$NINITIALIZE
.EXTRN DBG$NSYNTAX_ERROR
.EXTRN DBG$SET_STP_LVL
.EXTRN DBG$PSEUDO_PROG
.EXTRN LIB$SIGNAL, LIB$STOP
.EXTRN DBG$GB_CALL_NORMAL_RET
.EXTRN DBG$GB_RADIX, DBG$GB_LANGUAGE
.EXTRN DBG$GL_SIGN_FLAG
.EXTRN DBG$GV_CONTROL, DBG$GB_STP_PTR
.EXTRN DBG$GB_MOD_PTR, DBG$GL_CISREAD
.EXTRN DBG$GL_CIS_LEVELS
```



```
; Routine Size: 55 bytes,    Routine Base: DBG$CODE + 0000
```

```
463 0589 1 GLOBAL ROUTINE DBG$EVENT_SYNTAX(INPUT_DESC, VERB_NODE, MESSAGE_VECT) =
464 0590 1
465 0591 1 FUNCTION
466 0592 1 This routine parses a step, break, trace, or watch point command,
467 0593 1 after the '[SET] STEP', 'SET BREAK', 'SET TRACE', or 'SET WATCH'
468 0594 1 has been parsed. This routine is called from DBGNSET (or DBGNSTEP).
469 0595 1
470 0596 1 The following syntax is accepted:
471 0597 1
472 0598 1 SET [ BREAK : TRACE : WATCH ]
473 0599 1 [ /AFTER : <n> ]
474 0600 1 [ /TEMPORARY ]
475 0601 1 [ /[NO]SILENT ]
476 0602 1 <event>
477 0603 1 [ WHEN (<expression>) ]
478 0604 1 [ DO (<command list>) ]
479 0605 1
480 0606 1 If the event is a watchpoint, <event> is:
481 0607 1
482 0608 1 <address> [ , <address> ]*
483 0609 1
484 0610 1 otherwise, <event> is:
485 0611 1
486 0612 1 [ /READ : /WRITE : /MODIFY : /EXECUTE : /RETURN ]
487 0613 1 <address> [ , <address> ]* ;
488 0614 1 [ /EXCEPTION ] ;
489 0615 1 [ /[NO]SOURCE : /[NO]SYSTEM : /INTO : /OVER ]*
490 0616 1 [ /CALLS : /RETURN : /BRANCH : /LINE : /INSTRUCTION
491 0617 1 [ = [ <mnemonic> : ( <mnemonic> [ , <mnemonic> ] ) ] ]
492 0618 1 ]
493 0619 1
494 0620 1 SET STEP
495 0621 1 [ /[NO]SILENT
496 0622 1 [ /[NO]SOURCE : /[NO]SYSTEM : INTO : OVER ]*
497 0623 1 [ CALLS : RETURN : BRANCH : LINE : INSTRUCTION
498 0624 1 [ = [ <mnemonic> : ( <mnemonic> [ , <mnemonic> ] ) ] ]
499 0625 1 ]
500 0626 1
501 0627 1 STEP
502 0628 1 [ /[NO]SILENT
503 0629 1 [ /[NO]SOURCE : /[NO]SYSTEM : /INTO : /OVER ]*
504 0630 1 [ /CALLS : /RETURN : /BRANCH : /LINE : /INSTRUCTION
505 0631 1 [ = [ <mnemonic> : ( <mnemonic> [ , <mnemonic> ] ) ] ]
506 0632 1 ]
507 0633 1
508 0634 1
509 0635 1 INPUTS
510 0636 1 INPUT_DESC: The current command line input descriptor.
511 0637 1 VERB_NODE: A pointer to a DEBUG verb node, with the COMPOSITE
512 0638 1 attribute set to EVENT$K_SET_BREAK, EVENT$K_SET_TRACE,
513 0639 1 EVENT$K_SET_WATCH, EVENT$K_SET_STEP, or EVENT$K_STEP.
514 0640 1
515 0641 1 MESSAGE_VECT: The address of a longword to contain the address
516 0642 1 of a message argument vector.
517 0643 1
518 0644 1 OUTPUTS
519 0645 1 The parse tree headed by VERB_NODE is created to represent the
```



```

520 0646 1 1 1 command parsed. Errors cause this routine to return STSSK_SEVERE,
521 0647 1 1 1 otherwise STSSK_SUCCESS.
522 0648 1 1 1
523 0649 1 1 1
524 0650 2 2 2 BEGIN
525 0651 2 2 2
526 0652 2 2 2 MAP
527 0653 2 2 2 VERB_NODE: REF DBG$VERB_NODE; ! Verb node pointer
528 0654 2 2 2
529 0655 2 2 2 LOCAL
530 0656 2 2 2 STATUS, ! Returned status
531 0657 2 2 2 ADVERB_NODE: REF DBG$ADVERB_NODE, ! Adverb node pointer
532 0658 2 2 2 ADVERB_LINK: REF DBG$ADVERB_NODE, ! Adverb node link
533 0659 2 2 2 NOUN_NODE: REF DBG$NOUN_NODE, ! Noun node pointer
534 0660 2 2 2 NOUN_LINK: REF DBG$NOUN_NODE, ! Noun node link
535 0661 2 2 2 CMD_TYPE, ! Command type
536 0662 2 2 2 CMD_KIND, ! Command Kind
537 0663 2 2 2 SUB_KIND, ! Command sub-kind
538 0664 2 2 2 AFTER_COUNT, ! After count
539 0665 2 2 2 TEMPORARY, ! Temporary flag
540 0666 2 2 2 SILENT, ! Silent flag
541 0667 2 2 2 SOURCE, ! Source flag
542 0668 2 2 2 NOSILENT, ! Nosilent flag
543 0669 2 2 2 NOSOURCE, ! Nosource flag
544 0670 2 2 2 SYSTEM, ! System flag
545 0671 2 2 2 NOSYSTEM, ! Nosystem flag
546 0672 2 2 2 INTO, ! Into flag
547 0673 2 2 2 OVER; ! Over flag
548 0674 2 2 2
549 0675 2 2 2
550 0676 2 2 2 BIND
551 0677 2 2 2
552 0678 2 2 2
553 0679 2 2 2 ! Define the character strings to parse.
554 0680 2 2 2 !
555 0681 2 2 2 DBG$CS_AFTER = SAC ('AFTER'),
556 0682 2 2 2 DBG$CS_TEMPORARY = SAC ('TEMPORARY'),
557 0683 2 2 2 DBG$CS_SILENT = SAC ('SILENT'),
558 0684 2 2 2 DBG$CS_NOSILENT = SAC ('NOSILENT'),
559 0685 2 2 2 DBG$CS_WHEN = SAC ('WHEN'),
560 0686 2 2 2 DBG$CS_DO = SAC ('DO'),
561 0687 2 2 2 DBG$CS_READ = SAC ('READ'),
562 0688 2 2 2 DBG$CS_WRITE = SAC ('WRITE'),
563 0689 2 2 2 DBG$CS_MODIFY = SAC ('MODIFY'),
564 0690 2 2 2 DBG$CS_EXECUTE = SAC ('EXECUTE'),
565 0691 2 2 2 DBG$CS_RETURN = SAC ('RETURN'),
566 0692 2 2 2 DBG$CS_SOURCE = SAC ('SOURCE'),
567 0693 2 2 2 DBG$CS_NOSOURCE = SAC ('NOSOURCE'),
568 0694 2 2 2 DBG$CS_SYSTEM = SAC ('SYSTEM'),
569 0695 2 2 2 DBG$CS_NOSYSTEM = SAC ('NOSYSTEM'),
570 0696 2 2 2 DBG$CS_INT0 = SAC ('INT0'),
571 0697 2 2 2 DBG$CS_OVER = SAC ('OVER'),
572 0698 2 2 2 DBG$CS_EXCEPTION = SAC ('EXCEPTION'),
573 0699 2 2 2 DBG$CS_CALL = SAC ('CALLS'),
574 0700 2 2 2 DBG$CS_BRANCH = SAC ('BRANCH'),
575 0701 2 2 2 DBG$CS_LINE = SAC ('LINE'),
576 0702 2 2 2 DBG$CS_INSTRUCTION = SAC ('INSTRUCTION'),
```

```

577      0703      2      DBG$CS_EQUALS =      SAC ('='),
578      0704      2      DBG$CS_L_PAREN =      SAC ('('),
579      0705      2      DBG$CS_R_PAREN =      SAC (')'),
580      0706      2      DBG$CS_COMMA =      SAC (','),
581      0707      2      DBG$CS_SLASH =      SAC ('/'),
582      0708      2      DBG$CS_COLON =      SAC (':'),
583      0709      2      DBG$CS_CR =      UPLIT BYTE (1, DBG$K_CAR_RETURN);
584      0710
585      0711
586      0712
587      0713      ! Initialize the verb's adverb and noun pointers to null.
588      0714      !
589      0715      VERB_NODE [DBG$L_VERB_ADVERB_PTR] = 0;
590      0716      VERB_NODE [DBG$L_VERB_OBJECT_PTR] = 0;
591      0717
592      0718
593      0719      ! Initialize the adverb and noun links.
594      0720      !
595      0721      ADVERB_LINK = VERB_NODE [DBG$L_VERB_ADVERB_PTR];
596      0722      NOUN_LINK = VERB_NODE [DBG$L_VERB_OBJECT_PTR];
597      0723
598      0724
599      0725      ! Set the default command descriptors....
600      0726      !
601      0727      SELECTONE .VERB_NODE [DBG$B_VERB_COMPOSITE] OF
602      0728      SET
603      0729
604      0730
605      0731      [EVENT$K_SET_BREAK]:
606      0732      BEGIN
607      0733      CMD_TYPE = EVENT$K_TYPE_BREAK;
608      0734      CMD_KIND = EVENT$K_KIND_ACC;
609      0735      SUB_KIND = EVENT$K_ACC_EXEC;
610      0736      END;
611      0737
612      0738
613      0739      [EVENT$K_SET_TRACE]:
614      0740      BEGIN
615      0741      CMD_TYPE = EVENT$K_TYPE_TRACE;
616      0742      CMD_KIND = EVENT$K_KIND_ACC;
617      0743      SUB_KIND = EVENT$K_ACC_EXEC;
618      0744      END;
619      0745
620      0746
621      0747      [EVENT$K_SET_WATCH]:
622      0748      BEGIN
623      0749      CMD_TYPE = EVENT$K_TYPE_WATCH;
624      0750      CMD_KIND = EVENT$K_KIND_ACC;
625      0751      SUB_KIND = EVENT$K_ACC_MDFY;
626      0752      END;
627      0753
628      0754
629      0755      [EVENT$K_SET_STEP,EVENT$K_STEP]:
630      0756      BEGIN
631      0757      CMD_TYPE = EVENT$K_TYPE_STEPS;
632      0758      SUB_KIND = .DBG$GB_STP_PTR [EVENT$B_STEPPING_KIND];
633      0759      IF .SUB_KIND EQLU EVENT$K_ACC_RTRN

```



```

        THEN
            CMD_KIND = EVENT$K_KIND_ACC
        ELSE IF .SUB_KIND EQLU EVENT$K_EXC_EXC
        THEN
            CMD_KIND = EVENT$K_KIND_EXC
        ELSE
            CMD_KIND = EVENT$K_KIND_INS;
        END;
    TES;

    ! Setup the other default values.
    AFTER COUNT = 1;
    TEMPORARY = FALSE;
    SILENT = FALSE;
    NOSILENT = FALSE;
    SOURCE = FALSE;
    NOSOURCE = FALSE;
    SYSTEM = FALSE;
    NOSYSTEM = FALSE;
    INTO = FALSE;
    OVER = FALSE;
    IF .VERB_NODE [DBG$B_VERB_COMPOSITE] EQL EVENT$K_STEP
    THEN
        SILENT = NOT .DBG$GB_STP_PTR[EVENT$V_STEPPING_NOSILENT];

    ! Parse the command.  If this is not a SET STEP command, meaning that it
    ! is SET BREAK, SET TRACE, SET WATCH, or STEP, we look for qualifiers with
    ! "/" preceding.
    IF .VERB_NODE [DBG$B_VERB_COMPOSITE] NEQU EVENT$K_SET_STEP
    THEN
        BEGIN
            ! Look for the qualifiers....
            Parse the command according to the following syntax:
                [ /AFTER : <n> ]
                [ /[NO]SILENT ]
                [ /TEMPORARY ]
                [ /READ : /WRITE : /MODIFY : /EXECUTE : /RETURN ]
                  <address> [ , <address> ]*
                /EXCEPTION
                [ /[NO]SOURCE : /[NO]SYSTEM : /INTO : /OVER ]
                /CALLS
                /RETURN
                /BRANCH
                /INSTRUCTION [ = [ <mnemonic> : [ ( <mnemonic> [ , <mnemonic> ]* ) ] ] ]

```



```

: 691      0817      3      NOSILENT = TRUE;
: 692      0818      3      WHILE MATCH(DBG$CS_SLASH) DO
: 693      0819      4          BEGIN
: 694      0820      4              SELECTONE TRUE OF
: 695      0821      4                  SET
: 696      0822      4
: 697      0823      4
: 698      0824      4          ! Parse the /AFTER : <n> qualifier.
: 699      0825      4          !
: 700      0826      4          [MATCH (DBG$CS_AFTER)]:
: 701      0827      5              BEGIN
: 702      0828      5                  IF .CMD_TYPE EQLU EVENT$K_TYPE_BREAK OR
: 703      0829      5                      .CMD_TYPE EQLU EVENT$K_TYPE_TRACE OR
: 704      0830      5                      .CMD_TYPE EQLU EVENT$K_TYPE_WATCH
: 705      0831      5                  THEN
: 706      0832      6                      BEGIN
: 707      0833      6
: 708      0834      6                          ! Match the ':'.
: 709      0835      6                          !
: 710      0836      6                          NEED_MATCH (DBG$CS_COLON);
: 711      0837      6
: 712      0838      6                          ! Fill in the details.
: 713      0839      6                          !
: 714      0840      6                          IF NOT DBG$NSAVE_DECIMAL_INTEGER
: 715      0841      6                              (.INPUT_DESC,
: 716      0842      6                                  AFTER_COUNT,
: 717      0843      6                                  .MESSAGE_VECT
: 718      0844      6                              )
: 719      0845      6                          THEN
: 720      0846      6                              RETURN ST$K_SEVERE;
: 721      0847      6
: 722      0848      6                          END
: 723      0849      6
: 724      0850      6                      ELSE
: 725      0851      6                          SYNTAX_ERROR;
: 726      0852      5
: 727      0853      5                      END;
: 728      0854      5
: 729      0855      4
: 730      0856      4
: 731      0857      4          ! Parse the /TEMPORARY qualifier.
: 732      0858      4          !
: 733      0859      4          [MATCH (DBG$CS_TEMPORARY)]:
: 734      0860      4              BEGIN
: 735      0861      5
: 736      0862      5
: 737      0863      5
: 738      0864      5                  IF .CMD_TYPE EQLU EVENT$K_TYPE_BREAK OR
: 739      0865      5                      .CMD_TYPE EQLU EVENT$K_TYPE_TRACE OR
: 740      0866      5                      .CMD_TYPE EQLU EVENT$K_TYPE_WATCH
: 741      0867      5                  THEN
: 742      0868      6                      BEGIN
: 743      0869      6
: 744      0870      6                          ! Fill in the details.
: 745      0871      6                          !
: 746      0872      6                          TEMPORARY = TRUE;
: 747      0873      6
```


748 0874 6
749 0875 6
750 0876 5
751 0877 5
752 0878 5
753 0879 4
754 0880 4
755 0881 4
756 0882 4
757 0883 4
758 0884 4
759 0885 5
760 0886 5
761 0887 5
762 0888 4
763 0889 4
764 0890 4
765 0891 4
766 0892 4
767 0893 4
768 0894 5
769 0895 5
770 0896 5
771 0897 4
772 0898 4
773 0899 4
774 0900 4
775 0901 4
776 0902 4
777 0903 5
778 0904 5
779 0905 5
780 0906 5
781 0907 5
782 0908 5
783 0909 5
784 0910 5
785 0911 5
786 0912 6
787 0913 6
788 0914 6
789 0915 6
790 0916 6
791 0917 5
792 0918 5
793 0919 4
794 0920 4
795 0921 4
796 0922 4
797 0923 4
798 0924 4
799 0925 5
800 0926 5
801 0927 5
802 0928 5
803 0929 5
804 0930 5

```
END
ELSE
  SYNTAX_ERROR;
END;

! Parse the /SILENT qualifier.
[ MATCH (DBG$CS_SILENT, 2) ]:
  BEGIN
    SILENT = TRUE;
    NOSILENT = FALSE;
  END;

! Parse the /NOSILENT qualifier.
[ MATCH (DBG$CS_NOSILENT, 4) ]:
  BEGIN
    NOSILENT = TRUE;
    SILENT = FALSE;
  END;

! Parse /READ.
[ MATCH (DBG$CS_READ) ]:
  BEGIN

    ! Make sure that this is a SET BREAK or SET TRACE command,
    ! and then set the command kind to read access.
    IF .CMD_TYPE EQLU EVENT$K_TYPE_BREAK OR
       .CMD_TYPE EQLU EVENT$K_TYPE_TRACE
    THEN
      BEGIN
        CMD_KIND = EVENT$K_KIND_ACC;
        SUB_KIND = EVENT$K_ACC_READ;
      END
    ELSE
      SYNTAX_ERROR;
    END;

! Parse /WRITE.
[ MATCH (DBG$CS_WRITE) ]:
  BEGIN

    ! Make sure that this is a SET BREAK or SET TRACE command,
    ! and then set the command kind to write access.
```

```

: 805
: 806
: 807
: 808
: 809
: 810
: 811
: 812
: 813
: 814
: 815
: 816
: 817
: 818
: 819
: 820
: 821
: 822
: 823
: 824
: 825
: 826
: 827
: 828
: 829
: 830
: 831
: 832
: 833
: 834
: 835
: 836
: 837
: 838
: 839
: 840
: 841
: 842
: 843
: 844
: 845
: 846
: 847
: 848
: 849
: 850
: 851
: 852
: 853
: 854
: 855
: 856
: 857
: 858
: 859
: 860
: 861

```

```

0931
0932
0933
0934
0935
0936
0937
0938
0939
0940
0941
0942
0943
0944
0945
0946
0947
0948
0949
0950
0951
0952
0953
0954
0955
0956
0957
0958
0959
0960
0961
0962
0963
0964
0965
0966
0967
0968
0969
0970
0971
0972
0973
0974
0975
0976
0977
0978
0979
0980
0981
0982
0983
0984
0985
0986
0987

```

```

IF .CMD_TYPE EQLU EVENT$K_TYPE_BREAK OR
.CMD_TYPE EQLU EVENT$K_TYPE_TRACE
THEN
  BEGIN
    CMD_KIND = EVENT$K_KIND_ACC;
    SUB_KIND = EVENT$K_ACC_WRIT;
  END
ELSE
  SYNTAX_ERROR;
END;

! Parse /MODIFY.
[MATCH (DBG$CS_MODIFY)]:
  BEGIN

    ! Make sure that this is a SET BREAK or SET TRACE command,
    ! and then set the command kind to modify access.
    IF .CMD_TYPE EQLU EVENT$K_TYPE_BREAK OR
    .CMD_TYPE EQLU EVENT$K_TYPE_TRACE
    THEN
      BEGIN
        CMD_KIND = EVENT$K_KIND_ACC;
        SUB_KIND = EVENT$K_ACC_MDFY;
      END
    ELSE
      SYNTAX_ERROR;
    END;

! Parse /EXECUTE <address> [ , <address> ]*.
[MATCH (DBG$CS_EXECUTE)]:
  BEGIN

    ! Make sure that this is a SET BREAK or SET TRACE command,
    ! and then set the command kind to execute access.
    IF .CMD_TYPE EQLU EVENT$K_TYPE_BREAK OR
    .CMD_TYPE EQLU EVENT$K_TYPE_TRACE
    THEN
      BEGIN
        CMD_KIND = EVENT$K_KIND_ACC;
        SUB_KIND = EVENT$K_ACC_EXEC;
      END
    ELSE
      SYNTAX_ERROR;
    END;

```


862	0988	4
863	0989	4
864	0990	4
865	0991	4
866	0992	4
867	0993	5
868	0994	5
869	0995	5
870	0996	5
871	0997	5
872	0998	5
873	0999	5
874	1000	5
875	1001	6
876	1002	6
877	1003	6
878	1004	6
879	1005	6
880	1006	5
881	1007	5
882	1008	5
883	1009	4
884	1010	4
885	1011	4
886	1012	4
887	1013	4
888	1014	4
889	1015	5
890	1016	5
891	1017	5
892	1018	5
893	1019	5
894	1020	5
895	1021	5
896	1022	5
897	1023	5
898	1024	5
899	1025	6
900	1026	6
901	1027	6
902	1028	6
903	1029	6
904	1030	5
905	1031	5
906	1032	5
907	1033	4
908	1034	4
909	1035	4
910	1036	4
911	1037	4
912	1038	4
913	1039	5
914	1040	5
915	1041	5
916	1042	4
917	1043	4
918	1044	4

```
! Parse /RETURN <address> [ , <address> ]*.
```

```
[MATCH (DBG$CS_RETURN)]:
```

```
  BEGIN
```

```
    ! Make sure that this is not a SET WATCH command,  
    ! and then set the command kind to return access.
```

```
    IF .CMD_TYPE NEQU EVENT$K_TYPE_WATCH  
    THEN
```

```
      BEGIN
```

```
        CMD_KIND = EVENT$K_KIND_ACC;  
        SUB_KIND = EVENT$K_ACC_RTRN;  
      END
```

```
    ELSE  
      SYNTAX_ERROR;
```

```
  END;
```

```
! Parse /EXCEPTION.
```

```
[MATCH (DBG$CS_EXCEPTION)]:
```

```
  BEGIN
```

```
    ! Make sure that this is not a SET WATCH command,  
    ! and then set the command kind to execption.
```

```
    IF .CMD_TYPE EQLU EVENT$K_TYPE_BREAK OR  
       .CMD_TYPE EQLU EVENT$K_TYPE_TRACE OR  
       .CMD_TYPE EQLU EVENT$K_TYPE_STEPS
```

```
    THEN
```

```
      BEGIN
```

```
        CMD_KIND = EVENT$K_KIND_EXC;  
        SUB_KIND = EVENT$K_EXC_EXC;  
      END
```

```
    ELSE  
      SYNTAX_ERROR;
```

```
  END;
```

```
! Parse /SOURCE.
```

```
[MATCH (DBG$CS_SOURCE, 2)]:
```

```
  BEGIN
```

```
    SOURCE = TRUE;  
    NOSOURCE = FALSE;  
  END;
```

```
919 1045 4
920 1046 4
921 1047 4
922 1048 5
923 1049 5
924 1050 5
925 1051 4
926 1052 4
927 1053 4
928 1054 4
929 1055 4
930 1056 4
931 1057 5
932 1058 5
933 1059 5
934 1060 5
935 1061 5
936 1062 5
937 1063 5
938 1064 5
939 1065 6
940 1066 6
941 1067 6
942 1068 6
943 1069 6
944 1070 5
945 1071 5
946 1072 5
947 1073 4
948 1074 4
949 1075 4
950 1076 4
951 1077 4
952 1078 4
953 1079 5
954 1080 5
955 1081 5
956 1082 5
957 1083 5
958 1084 5
959 1085 5
960 1086 5
961 1087 6
962 1088 6
963 1089 6
964 1090 6
965 1091 6
966 1092 5
967 1093 5
968 1094 5
969 1095 4
970 1096 4
971 1097 4
972 1098 4
973 1099 4
974 1100 4
975 1101 5
```

```
! Parse /NOSOURCE.
[ MATCH (DBG$CS_NOSOURCE, 4) ]:
  BEGIN
  NOSOURCE = TRUE;
  SOURCE = FALSE;
  END;

! Parse /SYSTEM.
[ MATCH (DBG$CS_SYSTEM, 2) ]:
  BEGIN

  ! Make sure that this is not a SET WATCH command,
  ! and then set the command kind to call.
  IF .CMD_TYPE NEQU EVENT$K_TYPE_WATCH
  THEN
    BEGIN
    SYSTEM = TRUE;
    NOSYSTEM = FALSE;
    END
  ELSE
    SYNTAX_ERROR;
  END;

! Parse /NOSYSTEM.
[ MATCH (DBG$CS_NOSYSTEM, 4) ]:
  BEGIN

  ! Make sure that this is not a SET WATCH command,
  ! and then set the command kind to call.
  IF .CMD_TYPE NEQU EVENT$K_TYPE_WATCH
  THEN
    BEGIN
    NOSYSTEM = TRUE;
    SYSTEM = FALSE;
    END
  ELSE
    SYNTAX_ERROR;
  END;

! Parse /INTO.
[ MATCH (DBG$CS_INT0) ]:
  BEGIN
```



```
: 976 1102 5
: 977 1103 5
: 978 1104 5
: 979 1105 5
: 980 1106 5
: 981 1107 5
: 982 1108 5
: 983 1109 6
: 984 1110 6
: 985 1111 6
: 986 1112 6
: 987 1113 6
: 988 1114 5
: 989 1115 5
: 990 1116 4
: 991 1117 4
: 992 1118 4
: 993 1119 4
: 994 1120 4
: 995 1121 4
: 996 1122 5
: 997 1123 5
: 998 1124 5
: 999 1125 5
1000 1126 5
1001 1127 5
1002 1128 5
1003 1129 5
1004 1130 6
1005 1131 6
1006 1132 6
1007 1133 6
1008 1134 6
1009 1135 5
1010 1136 5
1011 1137 5
1012 1138 4
1013 1139 4
1014 1140 4
1015 1141 4
1016 1142 4
1017 1143 4
1018 1144 5
1019 1145 5
1020 1146 5
1021 1147 5
1022 1148 5
1023 1149 5
1024 1150 5
1025 1151 5
1026 1152 6
1027 1153 6
1028 1154 6
1029 1155 6
1030 1156 6
1031 1157 5
: 1032 1158 5
```

```
! Make sure that this is not a SET WATCH command,
! and then set the command kind to call.
IF .CMD_TYPE NEQU EVENT$K_TYPE_WATCH
THEN
    BEGIN
        INTO = TRUE;
        OVER = FALSE;
    END
ELSE
    SYNTAX_ERROR;
END;

! Parse /OVER.
[ MATCH (DBG$CS_OVER) ]:
    BEGIN

! Make sure that this is not a SET WATCH command,
! and then set the command kind to call.
IF .CMD_TYPE NEQU EVENT$K_TYPE_WATCH
THEN
    BEGIN
        OVER = TRUE;
        INTO = FALSE;
    END
ELSE
    SYNTAX_ERROR;
END;

! Parse /CALLS.
[ MATCH (DBG$CS_CALL) ]:
    BEGIN

! Make sure that this is not a SET WATCH command,
! and then set the command kind to call.
IF .CMD_TYPE NEQU EVENT$K_TYPE_WATCH
THEN
    BEGIN
        CMD_KIND = EVENT$K_KIND_INS;
        SUB_KIND = EVENT$K_INS_CALL;
    END
ELSE
    SYNTAX_ERROR;
```

```
: 1033 1159 5
: 1034 1160 4
: 1035 1161 4
: 1036 1162 4
: 1037 1163 4
: 1038 1164 4
: 1039 1165 4
: 1040 1166 5
: 1041 1167 5
: 1042 1168 5
: 1043 1169 5
: 1044 1170 5
: 1045 1171 5
: 1046 1172 5
: 1047 1173 5
: 1048 1174 6
: 1049 1175 6
: 1050 1176 6
: 1051 1177 6
: 1052 1178 6
: 1053 1179 5
: 1054 1180 5
: 1055 1181 5
: 1056 1182 4
: 1057 1183 4
: 1058 1184 4
: 1059 1185 4
: 1060 1186 4
: 1061 1187 4
: 1062 1188 5
: 1063 1189 5
: 1064 1190 5
: 1065 1191 5
: 1066 1192 5
: 1067 1193 5
: 1068 1194 5
: 1069 1195 5
: 1070 1196 6
: 1071 1197 6
: 1072 1198 6
: 1073 1199 6
: 1074 1200 6
: 1075 1201 5
: 1076 1202 5
: 1077 1203 5
: 1078 1204 4
: 1079 1205 4
: 1080 1206 4
: 1081 1207 4
: 1082 1208 4
: 1083 1209 4
: 1084 1210 4
: 1085 1211 5
: 1086 1212 5
: 1087 1213 5
: 1088 1214 5
: 1089 1215 5
```

```
END;

! Parse /BRANCH.
[ MATCH (DBG$CS_BRANCH) ]:
BEGIN

! Make sure that this is not a SET WATCH command,
! and then set the command kind to branch.
IF .CMD_TYPE NEQU EVENT$K_TYPE_WATCH
THEN
BEGIN
CMD_KIND = EVENT$K_KIND_INS;
SUB_KIND = EVENT$K_INS_BRAN;
END
ELSE
SYNTAX_ERROR;
END;

! Parse /LINE.
[ MATCH (DBG$CS_LINE) ]:
BEGIN

! Make sure that this is not a SET WATCH command,
! and then set the command kind to line.
IF .CMD_TYPE NEQU EVENT$K_TYPE_WATCH
THEN
BEGIN
CMD_KIND = EVENT$K_KIND_INS;
SUB_KIND = EVENT$K_INS_LINE;
END
ELSE
SYNTAX_ERROR;
END;

! Parse /INSTRUCTION
[ = [ <mnemonic> ; ( <mnemonic> [ , <mnemonic> ]* ) ] ].
[ MATCH (DBG$CS_INSTRUCTION) ]:
BEGIN

! Make sure that this is not a SET WATCH command,
! and then set the command kind to instruction.
```


1090	1216	5
1091	1217	5
1092	1218	5
1093	1219	6
1094	1220	6
1095	1221	6
1096	1222	6
1097	1223	6
1098	1224	6
1099	1225	6
1100	1226	7
1101	1227	6
1102	1228	7
1103	1229	7
1104	1230	7
1105	1231	7
1106	1232	7
1107	1233	7
1108	1234	7
1109	1235	7
1110	1236	7
1111	1237	8
1112	1238	7
1113	1239	7
1114	1240	7
1115	1241	7
1116	1242	7
1117	1243	8
1118	1244	8
1119	1245	9
1120	1246	9
1121	1247	9
1122	1248	9
1123	1249	9
1124	1250	9
1125	1251	9
1126	1252	9
1127	1253	9
1128	1254	9
1129	1255	9
1130	1256	8
1131	1257	8
1132	1258	8
1133	1259	8
1134	1260	8
1135	1261	8
1136	1262	8
1137	1263	8
1138	1264	7
1139	1265	8
1140	1266	8
1141	1267	8
1142	1268	8
1143	1269	8
1144	1270	8
1145	1271	8
1146	1272	8

```
!
IF .CMD_TYPE NEQU EVENT$K_TYPE_WATCH
THEN
  BEGIN
    CMD_KIND = EVENT$K_KIND_INS;
    SUB_KIND = EVENT$K_INS_EVERY;

    ! Look for a possible '='.
    !
    IF MATCH (DBG$CS_EQUALS)
    THEN
      BEGIN
        ! Set the kind to user defined instead of every.
        !
        SUB_KIND = EVENT$K_INS_USER;

        ! Collect mnemonic(s).
        !
        IF MATCH (DBG$CS_L_PAREN)
        THEN
          ! Collect mnemonic(s) and build an noun list.
          !
          BEGIN
            DO
              BEGIN
                ! Get a noun node for a <mnemonic>.
                !
                GET_NOUN_NODE;

                NOUN_NODE [DBG$L_NOUN_VALUE] =
                  DBG$OPCODE_INDEX(,INPUT_DESC);

              END
            WHILE (MATCH (DBG$CS_COMMA));

            ! Match the ')'.
            !
            NEED_MATCH (DBG$CS_R_PAREN);
          END
        ELSE
          BEGIN
            ! Get a noun node for a <mnemonic>.
            !
            GET_NOUN_NODE;

            NOUN_NODE [DBG$L_NOUN_VALUE] =
              DBG$OPCODE_INDEX(,INPUT_DESC);
```

```

: 1147      1273      7      END;
: 1148      1274      7
: 1149      1275      6      END;
: 1150      1276      6
: 1151      1277      6      END
: 1152      1278      6
: 1153      1279      5      ELSE
: 1154      1280      5      SYNTAX_ERROR;
: 1155      1281      5
: 1156      1282      4      END;
: 1157      1283      4
: 1158      1284      4
: 1159      1285      4      ! Check for ambiguous cases.
: 1160      1286      4      !
: 1161      1287      4      [MATCH(UPLIT BYTE(%ASCIC 'S'))]:
: 1162      1288      4      SIGNAL(DBG$_AMBIGQUAL, 1, UPLIT BYTE(%ASCIC 'S'));
: 1163      1289      4
: 1164      1290      4      [MATCH(UPLIT BYTE(%ASCIC 'NOS'), 3)]:
: 1165      1291      4      SIGNAL(DBG$_AMBIGQUAL, 1, UPLIT BYTE(%ASCIC 'NOS'));
: 1166      1292      4
: 1167      1293      4
: 1168      1294      4      ! Otherwise, we've got an undefined qualifier....
: 1169      1295      4      !
: 1170      1296      4      [OTHERWISE]:
: 1171      1297      4      SYNTAX_ERROR;
: 1172      1298      4
: 1173      1299      4      TES;
: 1174      1300      4
: 1175      1301      3      END;      ! End of WHILE loop over qualifiers
: 1176      1302      3
: 1177      1303      3      END      ! End of non-SET STEP command parse
: 1178      1304      3
: 1179      1305      3
: 1180      1306      3      ! Handle the SET STEP command qualifiers.
: 1181      1307      3      !
: 1182      1308      2      ELSE
: 1183      1309      2      BEGIN
: 1184      1310      2
: 1185      1311      2      ! Parse the SET STEP command according to the following syntax:
: 1186      1312      2      !
: 1187      1313      2      ! [ [NO]SILENT ]
: 1188      1314      2      ! [ [NO]SOURCE ; [NO]SYSTEM ; INTO ; OVER ]
: 1189      1315      2      !
: 1190      1316      2      ! CALLS
: 1191      1317      2      ! RETURN
: 1192      1318      2      ! BRANCH
: 1193      1319      2      ! INSTRUCTION [ = [ <mnemonic> ; [ ( <mnemonic> [ , <mnemonic> ]* ) ] ]
: 1194      1320      2      !
: 1195      1321      3      DO
: 1196      1322      4      BEGIN
: 1197      1323      4      SELECTONE TRUE OF
: 1198      1324      4      SET
: 1199      1325      4
: 1200      1326      4      ! Parse RETURN
: 1201      1327      4      !
: 1202      1328      4      [MATCH (DBG$CS_RETURN)]:
: 1203      1329      4
```


1204	1330	5	BEGIN
1205	1331	5	CMD_KIND = EVENT\$K_KIND_ACC;
1206	1332	5	SUB_KIND = EVENT\$K_ACC_RTRN;
1207	1333	4	END;
1208	1334	4	
1209	1335	4	
1210	1336	4	! Parse SOURCE.
1211	1337	4	!
1212	1338	4	[MATCH (DBG\$CS_SOURCE, 2)]:
1213	1339	5	BEGIN
1214	1340	5	SOURCE = TRUE;
1215	1341	5	NOSOURCE = FALSE;
1216	1342	4	END;
1217	1343	4	
1218	1344	4	
1219	1345	4	! Parse NOSOURCE.
1220	1346	4	!
1221	1347	4	[MATCH (DBG\$CS_NOSOURCE, 4)]:
1222	1348	5	BEGIN
1223	1349	5	NOSOURCE = TRUE;
1224	1350	5	SOURCE = FALSE;
1225	1351	4	END;
1226	1352	4	
1227	1353	4	
1228	1354	4	! Parse SYSTEM.
1229	1355	4	!
1230	1356	4	[MATCH (DBG\$CS_SYSTEM, 2)]:
1231	1357	5	BEGIN
1232	1358	5	SYSTEM = TRUE;
1233	1359	5	NOSYSTEM = FALSE;
1234	1360	4	END;
1235	1361	4	
1236	1362	4	
1237	1363	4	! Parse NOSYSTEM.
1238	1364	4	!
1239	1365	4	[MATCH (DBG\$CS_NOSYSTEM, 4)]:
1240	1366	5	BEGIN
1241	1367	5	NOSYSTEM = TRUE;
1242	1368	5	SYSTEM = FALSE;
1243	1369	4	END;
1244	1370	4	
1245	1371	4	
1246	1372	4	! Parse SILENT.
1247	1373	4	!
1248	1374	4	[MATCH (DBG\$CS_SILENT, 2)]:
1249	1375	5	BEGIN
1250	1376	5	SILENT = TRUE;
1251	1377	5	NOSILENT = FALSE;
1252	1378	4	END;
1253	1379	4	
1254	1380	4	
1255	1381	4	! Parse NOSILENT.
1256	1382	4	!
1257	1383	4	[MATCH (DBG\$CS_NOSILENT, 4)]:
1258	1384	5	BEGIN
1259	1385	5	SILENT = FALSE;
1260	1386	5	NOSILENT = TRUE;

1261	1387	4
1262	1388	4
1263	1389	4
1264	1390	4
1265	1391	4
1266	1392	4
1267	1393	5
1268	1394	5
1269	1395	5
1270	1396	4
1271	1397	4
1272	1398	4
1273	1399	4
1274	1400	4
1275	1401	4
1276	1402	5
1277	1403	5
1278	1404	5
1279	1405	4
1280	1406	4
1281	1407	4
1282	1408	4
1283	1409	4
1284	1410	4
1285	1411	5
1286	1412	5
1287	1413	5
1288	1414	4
1289	1415	4
1290	1416	4
1291	1417	4
1292	1418	4
1293	1419	4
1294	1420	5
1295	1421	5
1296	1422	5
1297	1423	4
1298	1424	4
1299	1425	4
1300	1426	4
1301	1427	4
1302	1428	4
1303	1429	5
1304	1430	5
1305	1431	5
1306	1432	4
1307	1433	4
1308	1434	4
1309	1435	4
1310	1436	4
1311	1437	4
1312	1438	5
1313	1439	5
1314	1440	5
1315	1441	4
1316	1442	4
1317	1443	4

```
END;

! Parse INTO.
[ MATCH (DBG$CS_INT0) ]:
  BEGIN
    INTO = TRUE;
    OVER = FALSE;
  END;

! Parse OVER.
[ MATCH (DBG$CS_OVER) ]:
  BEGIN
    OVER = TRUE;
    INTO = FALSE;
  END;

! Parse CALLS.
[ MATCH (DBG$CS_CALL) ]:
  BEGIN
    CMD_KIND = EVENT$K_KIND_INS;
    SUB_KIND = EVENT$K_INS_CALL;
  END;

! Parse BRANCH.
[ MATCH (DBG$CS_BRANCH) ]:
  BEGIN
    CMD_KIND = EVENT$K_KIND_INS;
    SUB_KIND = EVENT$K_INS_BRAN;
  END;

! Parse EXCEPTION
[ MATCH (DBG$CS_EXCEPTION) ]:
  BEGIN
    CMD_KIND = EVENT$K_KIND_EXC;
    SUB_KIND = EVENT$K_EXC_EXC;
  END;

! Parse LINE.
[ MATCH (DBG$CS_LINE) ]:
  BEGIN
    CMD_KIND = EVENT$K_KIND_INS;
    SUB_KIND = EVENT$K_INS_LINE;
  END;
```



```
: 1318      1444  4
: 1319      1445  4
: 1320      1446  4
: 1321      1447  4
: 1322      1448  5
: 1323      1449  5
: 1324      1450  5
: 1325      1451  5
: 1326      1452  5
: 1327      1453  5
: 1328      1454  5
: 1329      1455  6
: 1330      1456  5
: 1331      1457  6
: 1332      1458  6
: 1333      1459  6
: 1334      1460  6
: 1335      1461  6
: 1336      1462  6
: 1337      1463  6
: 1338      1464  6
: 1339      1465  7
: 1340      1466  6
: 1341      1467  6
: 1342      1468  6
: 1343      1469  6
: 1344      1470  7
: 1345      1471  7
: 1346      1472  8
: 1347      1473  8
: 1348      1474  8
: 1349      1475  8
: 1350      1476  8
: 1351      1477  8
: 1352      1478  8
: 1353      1479  8
: 1354      1480  8
: 1355      1481  8
: 1356      1482  7
: 1357      1483  7
: 1358      1484  7
: 1359      1485  7
: 1360      1486  7
: 1361      1487  7
: 1362      1488  7
: 1363      1489  7
: 1364      1490  6
: 1365      1491  7
: 1366      1492  7
: 1367      1493  7
: 1368      1494  7
: 1369      1495  7
: 1370      1496  7
: 1371      1497  7
: 1372      1498  6
: 1373      1499  6
: 1374      1500  5
```

```
! Parse INSTRUCTION
!   [ = [ <mnemonic> ; ( <mnemonic> [ , <mnemonic> ]* ) ] ].
[ MATCH (DBG$CS_INSTRUCTION) ]:
  BEGIN
    CMD_KIND = EVENT$K_KIND_INS;
    SUB_KIND = EVENT$K_INS_EVERY;

    ! Look for a possible '='.
    IF MATCH (DBG$CS_EQUALS)
    THEN
      BEGIN
        ! Set the kind to user defined instead of every.
        SUB_KIND = EVENT$K_INS_USER;

        ! Collect mnemonic(s).
        IF MATCH (DBG$CS_L_PAREN)
        THEN
          ! Collect mnemonic(s) and build an noun list.
          BEGIN
            DO
              BEGIN
                ! Get a noun node for a <mnemonic>.
                GET_NOUN_NODE;
                NOUN_NODE [DBG$L_NOUN_VALUE] =
                  DBG$OPCODE_INDEX( INPUT_DESC );
              END
            WHILE (MATCH (DBG$CS_COMMA));

            ! Match the ')'.
            NEED_MATCH (DBG$CS_R_PAREN);
          END
        ELSE
          BEGIN
            ! Get a noun node for a <mnemonic>.
            GET_NOUN_NODE;
            NOUN_NODE [DBG$L_NOUN_VALUE] =
              DBG$OPCODE_INDEX( INPUT_DESC );
          END;
        END;
      END;
    END;
```



```
: 1375      1501  5
: 1376      1502  4
: 1377      1503  4
: 1378      1504  4
: 1379      1505  4
: 1380      1506  4
: 1381      1507  4
: 1382      1508  4
: 1383      1509  4
: 1384      1510  4
: 1385      1511  4
: 1386      1512  4
: 1387      1513  4
: 1388      1514  4
: 1389      1515  4
: 1390      1516  4
: 1391      1517  4
: 1392      1518  4
: 1393      1519  4
: 1394      1520  4
: 1395      1521  4
: 1396      1522  4
: 1397      1523  3
: 1398      1524  3
: 1399      1525  2
: 1400      1526  2
: 1401      1527  2
: 1402      1528  2
: 1403      1529  2
: 1404      1530  2
: 1405      1531  2
: 1406      1532  2
: 1407      1533  2
: 1408      1534  2
: 1409      1535  2
: 1410      1536  2
: 1411      1537  2
: 1412      1538  2
: 1413      1539  2
: 1414      1540  3
: 1415      1541  3
: 1416      1542  3
: 1417      1543  3
: 1418      1544  3
: 1419      1545  3
: 1420      1546  3
: 1421      1547  3
: 1422      1548  3
: 1423      1549  3
: 1424      1550  3
: 1425      1551  3
: 1426      1552  3
: 1427      1553  2
: 1428      1554  2
: 1429      1555  2
: 1430      1556  2
: 1431      1557  2

      END;

      ! Check for ambiguous cases.
      [MATCH(UPLIT BYTE(%ASCIC 'S'))]:
        SIGNAL(DBG$_AMBIGQUAL, 1, UPLIT BYTE(%ASCIC 'S'));

      [MATCH(UPLIT BYTE(%ASCIC 'NOS'), 3)]:
        SIGNAL(DBG$_AMBIGQUAL, 1, UPLIT BYTE(%ASCIC 'NOS'));

      ! Otherwise, we've got an undefined qualifier (or something).
      [OTHERWISE]:
        SYNTAX_ERROR;

      TES;

      END

      WHILE (MATCH (DBG$CS_COMMA));

      END;                                ! End of SET STEP qualifier parsing

      ! Build up the adverb list based on what we've got so far.
      ! Note that if we've reached this point, we should have
      ! all qualifiers done, and the working variables reflect
      ! the command desired.

      ! Set up the command kind and possibly the sub-kind adverbs.

      GET ADVERB_NODE;
      SELECTONE :CMD_KIND OF
        SET
        [EVENT$K KIND ACC]:
          ADVERB_NODE [DBG$B ADVERB_LITERAL] =
            (SELECTONE .SUB_KIND OF
              SET
              [EVENT$K ACC READ]:
                ADVERB_READ;
              [EVENT$K ACC WRIT]:
                ADVERB_WRITE;
              [EVENT$K ACC MDIFY]:
                ADVERB_MODIFY;
              [EVENT$K ACC EXEC]:
                ADVERB_EXECUTE;
              [EVENT$K ACC RTRN]:
                ADVERB_RETURN;
              TES
            );

      [EVENT$K KIND EXC]:
        ADVERB_NODE [DBG$B ADVERB_LITERAL] = ADVERB_EXCEPTION;
```



```
: 1432      1558      2
: 1433      1559      2
: 1434      1560      2
: 1435      1561      2
: 1436      1562      2
: 1437      1563      2
: 1438      1564      2
: 1439      1565      2
: 1440      1566      2
: 1441      1567      2
: 1442      1568      2
: 1443      1569      2
: 1444      1570      2
: 1445      1571      2
: 1446      1572      2
: 1447      1573      2
: 1448      1574      2
: 1449      1575      2
: 1450      1576      2
: 1451      1577      2
: 1452      1578      2
: 1453      1579      2
: 1454      1580      2
: 1455      1581      2
: 1456      1582      2
: 1457      1583      2
: 1458      1584      2
: 1459      1585      2
: 1460      1586      2
: 1461      1587      2
: 1462      1588      2
: 1463      1589      2
: 1464      1590      2
: 1465      1591      2
: 1466      1592      2
: 1467      1593      2
: 1468      1594      2
: 1469      1595      2
: 1470      1596      2
: 1471      1597      2
: 1472      1598      2
: 1473      1599      2
: 1474      1600      2
: 1475      1601      2
: 1476      1602      2
: 1477      1603      2
: 1478      1604      2
: 1479      1605      2
: 1480      1606      2
: 1481      1607      2
: 1482      1608      2
: 1483      1609      2
: 1484      1610      2
: 1485      1611      2
: 1486      1612      2
: 1487      1613      2
: 1488      1614      2

[EVENT$K KIND INS]:
  ADVERB_NODE [DBG$B ADVERB_LITERAL] =
    (SELECTONE .SUB_KIND OF
      SET
        [EVENT$K INS CALL]:
          ADVERB_CALL;
        [EVENT$K INS BRAN]:
          ADVERB_BRANCH;
        [EVENT$K INS EVRY]:
          ADVERB_INSTRUCTION;
        [EVENT$K INS USER]:
          ADVERB_OPCODE;
        [EVENT$K INS LINE]:
          ADVERB_LINE;
      TES
    );
  TES;

! If the /AFTER, /UNTIL, and /EVERY are not at their defaults,
! setup an adverb node for the problem boys.
IF .AFTER_COUNT NEQU 1
THEN
  BEGIN
    GET_ADVERB_NODE;

    ! If the /AFTER count is 0, then this is really a kludge ah, I
    ! mean /TEMPORARY.
    IF .AFTER_COUNT EQLU 0
    THEN
      ADVERB_NODE [DBG$B ADVERB_LITERAL] = ADVERB_TEMPORARY
    ELSE
      BEGIN
        ADVERB_NODE [DBG$B ADVERB_LITERAL] = ADVERB_AFTER;
        ADVERB_NODE [DBG$L ADVERB_VALUE] = .AFTER_COUNT;
      END;
    END;

    ! If the /SILENT or /NOSILENT qualifier is present, make an Adverb Node
    ! for whichever one was specified last.
    IF .SILENT
    THEN
      BEGIN
        GET_ADVERB_NODE;
        ADVERB_NODE [DBG$B ADVERB_LITERAL] = ADVERB_SILENT;
      END
    ELSE IF .NOSILENT
    THEN
```



```

: 1489      1615      3      BEGIN
: 1490      1616      3      GET ADVERB_NODE;
: 1491      1617      3      ADVERB_NODE [DBG$B_ADVERB_LITERAL] = ADVERB_NOSILENT;
: 1492      1618      3      END;
: 1493      1619      3
: 1494      1620      3
: 1495      1621      3      ! If the /TEMPORARY qualifier is present, make an Adverb Node for it.
: 1496      1622      3      !
: 1497      1623      3      IF .TEMPORARY
: 1498      1624      3      THEN
: 1499      1625      3          BEGIN
: 1500      1626      3          GET ADVERB_NODE;
: 1501      1627      3          ADVERB_NODE [DBG$B_ADVERB_LITERAL] = ADVERB_TEMPORARY;
: 1502      1628      3          END;
: 1503      1629      3
: 1504      1630      3
: 1505      1631      3      ! If the /SOURCE or /NOSOURCE qualifier was present, make an Adverb Node
: 1506      1632      3      ! for it.
: 1507      1633      3      !
: 1508      1634      3      IF .SOURCE
: 1509      1635      3      THEN
: 1510      1636      3          BEGIN
: 1511      1637      3          GET ADVERB_NODE;
: 1512      1638      3          ADVERB_NODE [DBG$B_ADVERB_LITERAL] = ADVERB_SOURCE;
: 1513      1639      3          END;
: 1514      1640      3
: 1515      1641      3      IF .NOSOURCE
: 1516      1642      3      THEN
: 1517      1643      3          BEGIN
: 1518      1644      3          GET ADVERB_NODE;
: 1519      1645      3          ADVERB_NODE [DBG$B_ADVERB_LITERAL] = ADVERB_NOSOURCE;
: 1520      1646      3          END;
: 1521      1647      3
: 1522      1648      3
: 1523      1649      3      ! Handle /[NO]SYSTEM's, /INTO's or /OVER's.
: 1524      1650      3      !
: 1525      1651      3      IF .CMD_KIND EQLU EVENT$K_KIND INS OR
: 1526      1652      3      (.CMD_KIND EQLU EVENT$K_KIND ACC AND
: 1527      1653      3      .SUB_KIND EQLU EVENT$K_ACC_RTRN)
: 1528      1654      3      THEN
: 1529      1655      3          BEGIN
: 1530      1656      3          IF .SYSTEM
: 1531      1657      3          THEN
: 1532      1658      3              BEGIN
: 1533      1659      3              GET ADVERB_NODE;
: 1534      1660      3              ADVERB_NODE [DBG$B_ADVERB_LITERAL] = ADVERB_SYSTEM;
: 1535      1661      3              END;
: 1536      1662      3
: 1537      1663      3          IF .NOSYSTEM
: 1538      1664      3          THEN
: 1539      1665      3              BEGIN
: 1540      1666      3              GET ADVERB_NODE;
: 1541      1667      3              ADVERB_NODE [DBG$B_ADVERB_LITERAL] = ADVERB_NOSYSTEM;
: 1542      1668      3              END;
: 1543      1669      3
: 1544      1670      3          IF .INTO
: 1545      1671      3          THEN
```



```
: 1546      1672  4      BEGIN
: 1547      1673  4      GET ADVERB NODE;
: 1548      1674  4      ADVERB_NODE [DBG$B_ADVERB_LITERAL] = ADVERB_INT0;
: 1549      1675  3      END;
: 1550      1676  3
: 1551      1677  3      IF .OVER
: 1552      1678  3      THEN
: 1553      1679  4          BEGIN
: 1554      1680  4              GET ADVERB NODE;
: 1555      1681  4              ADVERB_NODE [DBG$B_ADVERB_LITERAL] = ADVERB_OVER;
: 1556      1682  3              END;
: 1557      1683  3
: 1558      1684  2          END;
: 1559      1685  2
: 1560      1686  2
: 1561      1687  2      ! If the command kind is an access kind, get the <address> [ , <address> ]*
: 1562      1688  2      ! as long as an <address> is terminated with a ', '.
: 1563      1689  2
: 1564      1690  2      IF .CMD_TYPE NEQU EVENT$K_TYPE_STEPS AND
: 1565      1691  2      .CMD_KIND EQLU EVENT$K_KIND_ACC
: 1566      1692  2      THEN
: 1567      1693  3          BEGIN
: 1568      1694  3              DO
: 1569      1695  4                  BEGIN
: 1570      1696  4
: 1571      1697  4
: 1572      1698  4                  ! Get a noun node for an <address>.
: 1573      1699  4                  !
: 1574      1700  4                  GET_NOUN_NODE;
: 1575      1701  4
: 1576      1702  4
: 1577      1703  4                  ! Get an <address> - the is a status warning
: 1578      1704  4                  ! if more is on the line. Surrounding this call, we
: 1579      1705  4                  ! turn on a flag saying we are picking up the address in
: 1580      1706  4                  ! a set break command. This is used in DBGPARSER to
: 1581      1707  4                  ! resolve potential ambiguities in SET BRE . DO (command)
: 1582      1708  4                  !
: 1583      1709  4                  DBG$GB_SET_BREAK_FLAG = TRUE;
: 1584      1710  4                  STATUS = DBG$NPARSE_ADDRESS(.INPUT_DESC,
: 1585      1711  4                      NOUN_NODE [DBG$L_NOUN_VALUE],
: 1586      1712  4                      .DBG$GB_RADIX[DBG$B_RADIX_INPUT],
: 1587      1713  4                      TOKEN$K_TERM_CMWHD0, .MESSAGE_VECT);
: 1588      1714  4                  DBG$GB_SET_BREAK_FLAG = FALSE;
: 1589      1715  4
: 1590      1716  4
: 1591      1717  4                  ! If the status indicates success, there is no
: 1592      1718  4                  ! more on the line, and we exit this loop.
: 1593      1719  4                  !
: 1594      1720  4                  IF .STATUS EQLU STS$K_SUCCESS
: 1595      1721  4                  THEN
: 1596      1722  4                      EXITLOOP;
: 1597      1723  4
: 1598      1724  4
: 1599      1725  4                  ! If the status is not a warning, something is
: 1600      1726  4                  ! wrong, so return with a failure.
: 1601      1727  4                  !
: 1602      1728  4                  IF .STATUS NEQU STS$K_WARNING
```

```

: 1603      1729 4      THEN
: 1604      1730 4      RETURN .STATUS
: 1605      1731 4
: 1606      1732 4      END
: 1607      1733 4      WHILE (MATCH (DBG$CS_COMMA));
: 1608      1734 4
: 1609      1735 4      END;
: 1610      1736 4
: 1611      1737 4
: 1612      1738 4      ! Check the input for a <cr> after parsing <address> [ , <address> ]*
: 1613      1739 4      ! or <mnemonic> [ , <mnemonic> ]. If there's nothing left on the
: 1614      1740 4      ! line we can leave.
: 1615      1741 4
: 1616      1742 4      IF MATCH (DBG$CS_CR)
: 1617      1743 4      THEN RETURN ST$K_SUCCESS;
: 1618      1744 4
: 1619      1745 4
: 1620      1746 4      ! We didn't match a <cr>, there's more on the line. If this is a STEP
: 1621      1747 4      ! command, get the step count....
: 1622      1748 4
: 1623      1749 4      IF .CMD_TYPE EQLU EVENT$K_TYPE_STEPS
: 1624      1750 4      THEN
: 1625      1751 4          IF .VERB_NODE [DBG$B_VERB_COMPOSITE] EQLU EVENT$K_STEP
: 1626      1752 4          THEN
: 1627      1753 4              BEGIN
: 1628      1754 4
: 1629      1755 4
: 1630      1756 4              ! Get an adverb node for a <step count>.
: 1631      1757 4              !
: 1632      1758 4              GET_ADVERB_NODE;
: 1633      1759 4
: 1634      1760 4
: 1635      1761 4              ! Get a <step count> - there is a status warning
: 1636      1762 4              ! if more is on the line.
: 1637      1763 4              !
: 1638      1764 4              ! Fill in the details.
: 1639      1765 4              !
: 1640      1766 4              ADVERB_NODE [DBG$B_ADVERB_LITERAL] = ADVERB_AFTER;
: 1641      1767 4              IF NOT DBG$NSAVE_DECIMAL_INTEGER
: 1642      1768 4                  (.INPUT_DESC,
: 1643      1769 4                  ADVERB_NODE [DBG$L_ADVERB_VALUE],
: 1644      1770 4                  .MESSAGE_VECT
: 1645      1771 4                  )
: 1646      1772 4              THEN
: 1647      1773 4                  RETURN ST$K_SEVERE
: 1648      1774 4              ELSE
: 1649      1775 4                  RETURN ST$K_SUCCESS;
: 1650      1776 4              END
: 1651      1777 4          ELSE
: 1652      1778 4              SYNTAX_ERROR;
: 1653      1779 4
: 1654      1780 4
: 1655      1781 4      ! This isn't a STEP command, but there's more on the line - maybe
: 1656      1782 4      ! a WHEN or a DO....
: 1657      1783 4
: 1658      1784 4      ! Parse a possible WHEN (<expression>).
: 1659      1785 4
```



```
: 1660      1786      3
: 1661      1787
: 1662      1788
: 1663      1789
: 1664      1790
: 1665      1791      | Match the '('.
: 1666      1792
: 1667      1793      NEED_MATCH (DBG$CS_L_PAREN);
: 1668      1794
: 1669      1795
: 1670      1796      | Get a new adverb node for the conditional expression.
: 1671      1797
: 1672      1798      GET_ADVERB_NODE;
: 1673      1799
: 1674      1800
: 1675      1801      | Flag a 'WHEN' and save the <expression>.
: 1676      1802
: 1677      1803      ADVERB_NODE [DBG$B_ADVERB_LITERAL] = ADVERB_WHEN;
: 1678      1804      DBG$NSAVE_BREAK_BUFFER (.INPUT_DESC, ADVERB_NODE [DBG$L_ADVERB_VALUE]);
: 1679      1805      END;
: 1680      1806
: 1681      1807
: 1682      1808      | Parse a Possible DO <command list>.
: 1683      1809
: 1684      1810      IF MATCH (DBG$CS_DO)
: 1685      1811      THEN
: 1686      1812      BEGIN
: 1687      1813
: 1688      1814
: 1689      1815      | Match the '('.
: 1690      1816
: 1691      1817      NEED_MATCH (DBG$CS_L_PAREN);
: 1692      1818
: 1693      1819
: 1694      1820      | Get a new adverb node for the command list.
: 1695      1821
: 1696      1822      GET_ADVERB_NODE;
: 1697      1823
: 1698      1824
: 1699      1825      | Flag a 'DO' and save the <command list>.
: 1700      1826
: 1701      1827      ADVERB_NODE [DBG$B_ADVERB_LITERAL] = ADVERB_DO;
: 1702      1828      DBG$NSAVE_BREAK_BUFFER (.INPUT_DESC, ADVERB_NODE [DBG$L_ADVERB_VALUE]);
: 1703      1829      RETURN ST$K_SUCCESS;
: 1704      1830      END;
: 1705      1831
: 1706      1832
: 1707      1833      | If there's more on the line, it's nothing we recognise. Signal
: 1708      1834      a syntax error.
: 1709      1835
: 1710      1836      IF MATCH (DBG$CS_CR) THEN RETURN ST$K_SUCCESS;
: 1711      1837      MESSAGE_VECT = DBG$NSYNTAX_ERROR (DBG$NNEXT_WORD (.INPUT_DESC));
: 1712      1838      RETURN ST$K_SEVERE;
: 1713      1839      END;
```

										.PSECT	DBG\$PLIT,NOWRT,	SHR,	PIC,0
59	52	41	52	52	45	54	46	41	05	00000	P.AAA:	.ASCII	<5>\AFTER\
			54	4F	50	4D	45	54	09	00006	P.AAB:	.ASCII	<9>\TEMPORARY\
				4E	45	4C	49	53	06	00010	P.AAC:	.ASCII	<6>\SILENT\
	54	4E	45	4C	49	53	4F	4E	08	00017	P.AAD:	.ASCII	<8>\NOSILENT\
					4E	45	48	57	04	00020	P.AAE:	.ASCII	<4>\WHEN\
							4F	44	02	00025	P.AAF:	.ASCII	<2>\DO\
					44	41	45	52	04	00028	P.AAG:	.ASCII	<4>\READ\
				45	54	49	52	57	05	0002D	P.AAH:	.ASCII	<5>\WRITE\
		45	59	46	49	44	4F	4D	06	00033	P.AAI:	.ASCII	<6>\MODIFY\
			54	55	43	45	58	45	07	0003A	P.AAJ:	.ASCII	<7>\EXECUTE\
			4E	52	55	54	45	52	06	00042	P.AAK:	.ASCII	<6>\RETURN\
			45	43	52	55	4F	53	06	00049	P.AAL:	.ASCII	<6>\SOURCE\
	45	43	52	55	4F	53	4F	4E	08	00050	P.AAM:	.ASCII	<8>\NOSOURCE\
			4D	45	54	53	59	53	06	00059	P.AAN:	.ASCII	<6>\SYSTEM\
					59	53	4F	4E	08	00060	P.AAO:	.ASCII	<8>\NOSYSTEM\
					4F	54	4E	49	04	00069	P.AAP:	.ASCII	<4>\INTO\
					52	45	56	4F	04	0006E	P.AAQ:	.ASCII	<4>\OVER\
	4E	4F	49	54	50	45	43	58	09	00073	P.AAR:	.ASCII	<9>\EXCEPTION\
					53	4C	4C	41	05	0007D	P.AAS:	.ASCII	<5>\CALLS\
				48	43	4E	41	52	06	00083	P.AAT:	.ASCII	<6>\BRANCH\
					45	4E	49	4C	04	0008A	P.AAU:	.ASCII	<4>\LINE\
4E	4F	49	54	43	55	52	54	53	0B	0008F	P.AAV:	.ASCII	<11>\INSTRUCTION\
								3D	01	0009B	P.AAW:	.ASCII	<1>\=\
								28	01	0009D	P.AAX:	.ASCII	<1>\(\
								29	01	0009F	P.AAY:	.ASCII	<1>\)\
								2C	01	000A1	P.AAZ:	.ASCII	<1>\,\
								2F	01	000A3	P.ABA:	.ASCII	<1>\/\
								3A	01	000A5	P.ABB:	.ASCII	<1>\:\
								0D	01	000A7	P.ABC:	.BYTE	1, 13
								53	01	000A9	P.ABD:	.ASCII	<1>\S\
								53	01	000AB	P.ABE:	.ASCII	<1>\S\
					53	4F	4E	03	000AD	P.ABF:	.ASCII	<3>\NOS\	
					53	4F	4E	03	000B1	P.ABG:	.ASCII	<3>\NOS\	
								53	01	000B5	P.ABH:	.ASCII	<1>\S\
								53	01	000B7	P.ABI:	.ASCII	<1>\S\
					53	4F	4E	03	000B9	P.ABJ:	.ASCII	<3>\NOS\	
					53	4F	4E	03	000BD	P.ABK:	.ASCII	<3>\NOS\	

DBG\$CS_AFTER=	P.AAA
DBG\$CS_TEMPORARY=	P.AAB
DBG\$CS_SILENT=	P.AAC
DBG\$CS_NOSILENT=	P.AAD
DBG\$CS_WHEN=	P.AAE
DBG\$CS_DO=	P.AAF
DBG\$CS_READ=	P.AAG
DBG\$CS_WRITE=	P.AAH
DBG\$CS_MODIFY=	P.AAI
DBG\$CS_EXECUTE=	P.AAJ
DBG\$CS_RETURN=	P.AAK
DBG\$CS_SOURCE=	P.AAL
DBG\$CS_NOSOURCE=	P.AAM
DBG\$CS_SYSTEM=	P.AAN
DBG\$CS_NOSYSTEM=	P.AAO
DBG\$CS_INT0=	P.AAP
DBG\$CS_OVER=	P.AAQ

DBG\$CS_EXCEPTION= P.AAR
DBG\$CS_CALL= P.AAS
DBG\$CS_BRANCH= P.AAT
DBG\$CS_LINE= P.AAU
DBG\$CS_INSTRUCTION= P.AAV
DBG\$CS_EQUALS= P.AAW
DBG\$CS_L_PAREN= P.AAX
DBG\$CS_R_PAREN= P.AAY
DBG\$CS_COMMA= P.AAZ
DBG\$CS_SLASH= P.ABA
DBG\$CS_COLON= P.ABB
DBG\$CS_CR= P.ABC

```
.PSECT  DBG$CODE,NOWRT,  SHR,  PIC,0

OFFC 00000

.ENTRY  DBG$EVENT_SYNTAX, Save R2,R3,R4,R5,R6,R7,-
      R8,R9,R10,R11
5E      24  C2 00002  SUBL2  #36, SP
50      08  AC D0 00005  MOVL  VERB_NODE, R0
      04  A0 7C 00009  CLRQ   4(R0)
58      04  A0 9E 0000C  MOVAB  4(R0), ADVERB_LINK
59      08  A0 9E 00010  MOVAB  8(R0), NOUN_LINK
50      01  A0 9A 00014  MOVZBL 1(R0), R0
01      50  91 00018  CMPB   R0, #1
      04  12 0001B  BNEQ   1$
      56  D4 0001D  CLRL   CMD_TYPE
      08  11 0001F  BRB    2$
0E      50  91 00021 1$:  CMPB   R0, #14
      0A  12 00024  BNEQ   3$
56      01  D0 00026  MOVL   #1, CMD_TYPE
      57  D4 00029 2$:  CLRL   CMD_KIND
55      03  D0 0002B  MOVL   #3, SUB_KIND
      3A  11 0002E  BRB    7$
11      50  91 00030 3$:  CMPB   R0, #17
      08  12 00033  BNEQ   4$
56      02  7D 00035  MOVQ   #2, CMD_TYPE
55      02  D0 00038  MOVL   #2, SUB_KIND
      2D  11 0003B  BRB    7$
0B      50  91 0003D 4$:  CMPB   R0, #11
      28  1F 00040  BLSSU  7$
0C      50  91 00042  CMPB   R0, #12
      23  1A 00045  BGTRU  7$
56      03  D0 00047  MOVL   #3, CMD_TYPE
51      00  D0 0004A  MOVL  DBG$GB_STP_PTR, R1
55      61  9A 00051  MOVZBL (R1), SUB_KIND
04      55  D1 00054  CMPL   SUB_KIND, #4
      04  12 00057  BNEQ   5$
      57  D4 00059  CLRL   CMD_KIND
      0D  11 0005B  BRB    7$
0F      55  D1 0005D 5$:  CMPL   SUB_KIND, #15
      05  12 00060  BNEQ   6$
57      02  D0 00062  MOVL   #2, CMD_KIND
      03  11 00065  BRB    7$
      57  01  D0 00067 6$:  MOVL   #1, CMD_KIND
20      AE  01  D0 0006A 7$:  MOVL   #1, AFTER_COUNT
      5A  7C 0006E  CLRQ   NOSILENT
```

5B

61

	14	AE	7C	00070	CLRQ	SOURCE	0781
	0C	AE	7C	00073	CLRQ	SYSTEM	0783
	04	AE	7C	00076	CLRQ	INTO	0785
		6E	D4	00079	CLRL	OVER	0786
	1C	AE	D4	0007B	CLRL	28(SP)	0787
0C		50	91	0007E	CMPB	R0, #12	
		12	12	00081	BNEQ	8\$	
	1C	AE	D6	00083	INCL	28(SP)	
51	00000000G	00	D0	00086	MOVL	DBG\$GB_STP_PTR, R1	0789
01		0C	EF	0008D	EXTZV	#12, #T, (R1), SILENT	
5B		5B	D2	00092	MCOML	SILENT, SILENT	
53		04	AC	D0	00095	8\$: MOVL	0818
0B			50	91	00099	INPUT_DESC, R3	0796
			03	12	0009C	CMPB	
		04	04	31	0009E	R0, #T1	
5A			01	D0	000A1	BNEQ	
			01	DD	000A4	9\$: BRW	0817
	00000000'		53	DD	000AC	10\$: MOVL	0818
			03	FB	000AE	#1, NOSILENT	
00000000G	00		50	E8	000B5	PUSHL	
			06	82	31	DBG\$CS_SLASH	
			01	DD	000BB	R3	
	00000000'		53	DD	000C3	#3, DBG\$NMATCH	
			03	FB	000C5	R0, 11\$	
00000000G	00		50	D1	000CC	103\$	
			51	12	000CF	PUSHL	0826
			56	D5	000D1	DBG\$CS_AFTER	
			0A	13	000D3	R3	
			56	D1	000D5	#3, DBG\$NMATCH	
01			05	13	000D8	R0, #1	
			56	D1	000DA	15\$	
02			65	12	000DD	CMD_TYPE	0828
			01	DD	000DF	12\$: BEQL	
	00000000'		53	DD	000E7	12\$: CMPL	0829
			03	FB	000E9	12\$: BEQL	
00000000G	00		50	E8	000F0	12\$: CMPL	0830
			01	DD	000F3	16\$: BNEQ	
	00000000'		53	DD	000FB	PUSHL	0837
			03	FB	000FD	#1	
00000000G	00		50	E8	00104	DBG\$CS_COLON	
			09	90	31	R3	
			09	3F	31	#3, DBG\$NMATCH	
			0C	AC	DD	R0, 14\$	
	24		53	DD	00113	R0, 14\$	
			03	FB	00115	#1	
00000000G	00		50	E8	0011C	DBG\$CS_CR	
			09	8E	31	R3	
			01	DD	00122	#3, DBG\$NMATCH	
	00000000'		53	DD	0012A	R0, 13\$	
			03	FB	0012C	143\$	
00000000G	00		50	D1	00133	139\$	
						MESSAGE_VECT	0845
						AFTER_COUNT	0843
						R3	
						#3, DBG\$NSAVE_DECIMAL_INTEGER	
						R0, 10\$	
						145\$	
						#1	0848
						DBG\$CS_TEMPORARY	0860
						R3	
						#3, DBG\$NMATCH	
						R0, #1	

		14	12	00136	BNEQ	18\$		
		56	D5	00138	TSTL	CMD_TYPE		0864
		0A	13	0013A	BEQL	17\$		
	01	56	D1	0013C	CMPL	CMD_TYPE, #1		0865
		05	13	0013F	BEQL	17\$		
	02	56	D1	00141	CMPL	CMD_TYPE, #2		0866
		5B	12	00144	BNEQ	21\$		
18	AE	01	D0	00146	MOVL	#1, TEMPORARY		0873
		5B	11	0014A	BRB	23\$		0864
		02	DD	0014C	PUSHL	#2		0884
	00000000'	EF	9F	0014E	PUSHAB	DBG\$CS_SILENT		
		53	DD	00154	PUSHL	R3		
00000000G	00	03	FB	00156	CALLS	#3, DBG\$NMATCH		
	01	50	D1	0015D	CMPL	R0, #1		
		07	12	00160	BNEQ	19\$		
	5B	01	D0	00162	MOVL	#1, SILENT		0886
		5A	D4	00165	CLRL	NOSILENT		0887
		64	11	00167	BRB	26\$		0820
		04	DD	00169	PUSHL	#4		0893
	00000000'	EF	9F	0016B	PUSHAB	DBG\$CS_NOSILENT		
		53	DD	00171	PUSHL	R3		
00000000G	00	03	FB	00173	CALLS	#3, DBG\$NMATCH		
	01	50	D1	0017A	CMPL	R0, #1		
		05	12	0017D	BNEQ	20\$		
	5A	01	7D	0017F	MOVQ	#1, NOSILENT		0895
		6F	11	00182	BRB	29\$		0820
		01	DD	00184	PUSHL	#1		0902
	00000000'	EF	9F	00186	PUSHAB	DBG\$CS_READ		
		53	DD	0018C	PUSHL	R3		
00000000G	00	03	FB	0018E	CALLS	#3, DBG\$NMATCH		
	01	50	D1	00195	CMPL	R0, #1		
		0F	12	00198	BNEQ	24\$		
		56	D5	0019A	TSTL	CMD_TYPE		0909
		05	13	0019C	BEQL	22\$		
	01	56	D1	0019E	CMPL	CMD_TYPE, #1		0910
		6F	12	001A1	BNEQ	31\$		
		57	D4	001A3	CLRL	CMD_KIND		0913
		55	D4	001A5	CLRL	SUB_KIND		0914
		70	11	001A7	BRB	33\$		0909
		01	DD	001A9	PUSHL	#1		0924
	00000000'	EF	9F	001AB	PUSHAB	DBG\$CS_WRITE		
		53	DD	001B1	PUSHL	R3		
00000000G	00	03	FB	001B3	CALLS	#3, DBG\$NMATCH		
	01	50	D1	001BA	CMPL	R0, #1		
		10	12	001BD	BNEQ	27\$		
		56	D5	001BF	TSTL	CMD_TYPE		0931
		05	13	001C1	BEQL	25\$		
	01	56	D1	001C3	CMPL	CMD_TYPE, #1		0932
		6E	12	001C6	BNEQ	35\$		
		57	D4	001C8	CLRL	CMD_KIND		0935
	55	01	D0	001CA	MOVL	#1, SUB_KIND		0936
		6F	11	001CD	BRB	37\$		0931
		01	DD	001CF	PUSHL	#1		0946
	00000000'	EF	9F	001D1	PUSHAB	DBG\$CS_MODIFY		
		53	DD	001D7	PUSHL	R3		
00000000G	00	03	FB	001D9	CALLS	#3, DBG\$NMATCH		
	01	50	D1	001E0	CMPL	R0, #1		

		10	12	001E3	BNEQ	30\$		
		56	D5	001E5	TSTL	CMD_TYPE		0953
		05	13	001E7	BEQL	28\$		
	01	56	D1	001E9	CMPL	CMD_TYPE, #1		0954
		48	12	001EC	BNEQ	35\$		
		57	D4	001EE	CLRL	CMD_KIND		0957
	55	02	D0	001F0	MOVL	#2, SUB_KIND		0958
		75	11	001F3	BRB	40\$		0953
		01	DD	001F5	PUSHL	#1		0969
		EF	9F	001F7	PUSHAB	DBG\$CS_EXECUTE		
		53	DD	001FD	PUSHL	R3		
00000000G	00	03	FB	001FF	CALLS	#3, DBG\$NMATCH		
	01	50	D1	00206	CMPL	R0, #1		
		10	12	00209	BNEQ	34\$		
		56	D5	0020B	TSTL	CMD_TYPE		0976
		05	13	0020D	BEQL	32\$		
	01	56	D1	0020F	CMPL	CMD_TYPE, #1		0977
		22	12	00212	BNEQ	35\$		
		57	D4	00214	CLRL	CMD_KIND		0980
	55	03	D0	00216	MOVL	#3, SUB_KIND		0981
		6E	11	00219	BRB	42\$		0976
		01	DD	0021B	PUSHL	#1		0992
		EF	9F	0021D	PUSHAB	DBG\$CS_RETURN		
		53	DD	00223	PUSHL	R3		
00000000G	00	03	FB	00225	CALLS	#3, DBG\$NMATCH		
	01	50	D1	0022C	CMPL	R0, #1		
		0F	12	0022F	BNEQ	38\$		
	02	56	D1	00231	CMPL	CMD_TYPE, #2		0999
		03	12	00234	BNEQ	36\$		
		07FF	31	00236	BRW	138\$		
		57	D4	00239	CLRL	CMD_KIND		1002
	55	04	D0	0023B	MOVL	#4, SUB_KIND		1003
		65	11	0023E	BRB	44\$		0999
		01	DD	00240	PUSHL	#1		1014
		EF	9F	00242	PUSHAB	DBG\$CS_EXCEPTION		
		53	DD	00248	PUSHL	R3		
00000000G	00	03	FB	0024A	CALLS	#3, DBG\$NMATCH		
	01	50	D1	00251	CMPL	R0, #1		
		16	12	00254	BNEQ	41\$		
		56	D5	00256	TSTL	CMD_TYPE		1021
		0A	13	00258	BEQL	39\$		
	01	56	D1	0025A	CMPL	CMD_TYPE, #1		1022
		05	13	0025D	BEQL	39\$		
	03	56	D1	0025F	CMPL	CMD_TYPE, #3		1023
		D2	12	00262	BNEQ	35\$		
	57	02	D0	00264	MOVL	#2, CMD_KIND		1026
	55	0F	D0	00267	MOVL	#15, SUB_KIND		1027
		7E	11	0026A	BRB	47\$		1021
		02	DD	0026C	PUSHL	#2		1038
		EF	9F	0026E	PUSHAB	DBG\$CS_SOURCE		
		53	DD	00274	PUSHL	R3		
00000000G	00	03	FB	00276	CALLS	#3, DBG\$NMATCH		
	01	50	D1	0027D	CMPL	R0, #1		
		09	12	00280	BNEQ	43\$		
14	AE	01	D0	00282	MOVL	#1, SOURCE		1040
		AE	D4	00286	CLRL	NOSOURCE		1041
		5F	11	00289	BRB	47\$		0820

		00000000'	04 DD 0028B 43\$:	PUSHL #4		1047
			EF 9F 0028D	PUSHAB	DBG\$CS_NOSOURCE	
00000000G	00		53 DD 00293	PUSHL R3		
	01		03 FB 00295	CALLS #3, DBG\$NMATCH		
			50 D1 0029C	CMPL R0, #1		
			06 12 0029F	BNEQ 45\$		
10	AE		01 7D 002A1	MOVQ #1, NOSOURCE		1049
			66 11 002A5 44\$:	BRB 49\$		0820
			02 DD 002A7 45\$:	PUSHL #2		1056
		00000000'	EF 9F 002A9	PUSHAB	DBG\$CS_SYSTEM	
			53 DD 002AF	PUSHL R3		
00000000G	00		03 FB 002B1	CALLS #3, DBG\$NMATCH		
	01		50 D1 002B8	CMPL R0, #1		
			0E 12 002BB	BNEQ 46\$		
	02		56 D1 002BD	CMPL CMD_TYPE, #2		1063
			66 13 002C0	BEQL 51\$		
0C	AE		01 D0 002C2	MOVL #1, SYSTEM		1066
		08	AE D4 002C6	CLRL NOSYSTEM		1067
			62 11 002C9	BRB 52\$		1063
			04 DD 002CB 46\$:	PUSHL #4		1078
		00000000'	EF 9F 002CD	PUSHAB	DBG\$CS_NOSYSTEM	
			53 DD 002D3	PUSHL R3		
00000000G	00		03 FB 002D5	CALLS #3, DBG\$NMATCH		
	01		50 D1 002DC	CMPL R0, #1		
			0B 12 002DF	BNEQ 48\$		
	02		56 D1 002E1	CMPL CMD_TYPE, #2		1085
			62 13 002E4	BEQL 54\$		
08	AE		01 7D 002E6	MOVQ #1, NOSYSTEM		1088
			64 11 002EA 47\$:	BRB 55\$		1085
			01 DD 002EC 48\$:	PUSHL #1		1100
		00000000'	EF 9F 002EE	PUSHAB	DBG\$CS_INT0	
			53 DD 002F4	PUSHL R3		
00000000G	00		03 FB 002F6	CALLS #3, DBG\$NMATCH		
	01		50 D1 002FD	CMPL R0, #1		
			0D 12 00300	BNEQ 50\$		
	02		56 D1 00302	CMPL CMD_TYPE, #2		1107
			64 13 00305	BEQL 57\$		
04	AE		01 D0 00307	MOVL #1, INT0		1110
			6E D4 0030B	CLRL OVER		1111
			64 11 0030D 49\$:	BRB 58\$		1107
			01 DD 0030F 50\$:	PUSHL #1		1121
		00000000'	EF 9F 00311	PUSHAB	DBG\$CS_OVER	
			53 DD 00317	PUSHL R3		
00000000G	00		03 FB 00319	CALLS #3, DBG\$NMATCH		
	01		50 D1 00320	CMPL R0, #1		
			0A 12 00323	BNEQ 53\$		
	02		56 D1 00325	CMPL CMD_TYPE, #2		1128
			64 13 00328 51\$:	BEQL 60\$		
	6E		01 7D 0032A	MOVQ #1, OVER		1131
			67 11 0032D 52\$:	BRB 61\$		1128
			01 DD 0032F 53\$:	PUSHL #1		1143
		00000000'	EF 9F 00331	PUSHAB	DBG\$CS_CALL	
			53 DD 00337	PUSHL R3		
00000000G	00		03 FB 00339	CALLS #3, DBG\$NMATCH		
	01		50 D1 00340	CMPL R0, #1		
			0D 12 00343	BNEQ 56\$		
	02		56 D1 00345	CMPL CMD_TYPE, #2		1150

		68	13	00348	54\$:	BEQL	64\$		
	57	01	D0	0034A		MOVL	#1, CMD_KIND		1153
	55	05	D0	0034D		MOVL	#5, SUB_KIND		1154
		44	11	00350	55\$:	BRB	61\$		1150
		01	DD	00352	56\$:	PUSHL	#1		1165
		EF	9F	00354		PUSHAB	DBG\$CS_BRANCH		
		53	DD	0035A		PUSHL	R3		
00000000G	00	03	FB	0035C		CALLS	#3, DBG\$NMATCH		
	01	50	D1	00363		CMPL	R0, #1		
		0D	12	00366		BNEQ	59\$		
	02	56	D1	00368		CMPL	CMD_TYPE, #2		1172
		48	13	0036B	57\$:	BEQL	64\$		
	57	01	D0	0036D		MOVL	#1, CMD_KIND		1175
	55	06	D0	00370		MOVL	#6, SUB_KIND		1176
		21	11	00373	58\$:	BRB	61\$		1172
		01	DD	00375	59\$:	PUSHL	#1		1187
		EF	9F	00377		PUSHAB	DBG\$CS_LINE		
		53	DD	0037D		PUSHL	R3		
00000000G	00	03	FB	0037F		CALLS	#3, DBG\$NMATCH		
	01	50	D1	00386		CMPL	R0, #1		
		0E	12	00389		BNEQ	62\$		
	02	56	D1	0038B		CMPL	CMD_TYPE, #2		1194
		25	13	0038E	60\$:	BEQL	64\$		
	57	01	D0	00390		MOVL	#1, CMD_KIND		1197
	55	07	D0	00393		MOVL	#7, SUB_KIND		1198
		FD0B	31	00396	61\$:	BRW	10\$		1194
		01	DD	00399	62\$:	PUSHL	#1		1210
		EF	9F	0039B		PUSHAB	DBG\$CS_INSTRUCTION		
		53	DD	003A1		PUSHL	R3		
00000000G	00	03	FB	003A3		CALLS	#3, DBG\$NMATCH		
	01	50	D1	003AA		CMPL	R0, #1		
		03	13	003AD		BEQL	63\$		
		00A7	31	003AF		BRW	68\$		
	02	56	D1	003B2	63\$:	CMPL	CMD_TYPE, #2		1217
		7B	13	003B5	64\$:	BEQL	66\$		
	57	01	D0	003B7		MOVL	#1, CMD_KIND		1220
	55	08	D0	003BA		MOVL	#8, SUB_KIND		1221
		01	DD	003BD		PUSHL	#1		1226
		EF	9F	003BF		PUSHAB	DBG\$CS_EQUALS		
		53	DD	003C5		PUSHL	R3		
00000000G	00	03	FB	003C7		CALLS	#3, DBG\$NMATCH		
	C5	50	E9	003CE		BLBC	R0, 61\$		
	55	09	D0	003D1		MOVL	#9, SUB_KIND		1232
		01	DD	003D4		PUSHL	#1		1237
		EF	9F	003D6		PUSHAB	DBG\$CS_L_PAREN		
		53	DD	003DC		PUSHL	R3		
00000000G	00	03	FB	003DE		CALLS	#3, DBG\$NMATCH		
	4D	50	E9	003E5		BLBC	R0, 67\$		
		04	DD	003E8	65\$:	PUSHL	#4		1245
00000000G	00	01	FB	003EA		CALLS	#1, DBG\$GET_TEMPMEM		
	54	50	D0	003F1		MOVL	R0, NOUN_NODE		
		08	A4	003F4		CLRL	8(NOUN_NODE)		
	69	54	D0	003F7		MOVL	NOUN_NODE, (NOUN_LINK)		
	59	08	A4	003FA		MOVAB	8(NOUN_NODE), NOON_LINK		
		53	DD	003FE		PUSHL	R3		1253
00000000G	00	01	FB	00400		CALLS	#1, DBG\$OPCODE_INDEX		
	64	50	D0	00407		MOVL	R0, (NOUN_NODE)		

		00000000'	01 DD 0040A	PUSHL #1	1256
			EF 9F 0040C	PUSHAB DBG\$CS_COMMA	
00000000G	00		53 DD 00412	PUSHL R3	
	CA		03 FB 00414	CALLS #3, DBG\$NMATCH	
			50 E8 0041B	BLBS R0, 65\$	
		00000000'	01 DD 0041E	PUSHL #1	1261
			EF 9F 00420	PUSHAB DBG\$CS_R_PAREN	
			53 DD 00426	PUSHL R3	
00000000G	00		03 FB 00428	CALLS #3, DBG\$NMATCH	
	70		50 E8 0042F	BLBS R0, 71\$	
		0603	31 00432 66\$:	BRW 138\$	
			04 DD 00435 67\$:	PUSHL #4	1265
00000000G	00		01 FB 00437	CALLS #1, DBG\$GET_TEMP MEM	
	54		50 D0 0043E	MOVL R0, NOUN_NODE	
		08	A4 D4 00441	CLRL 8(NOUN_NODE)	
	69		54 D0 00444	MOVL NOUN_NODE, (NOUN_LINK)	
	59	08	A4 9E 00447	MOVAB 8(NOUN_NODE), NOUN_LINK	
			53 DD 0044B	PUSHL R3	1272
00000000G	00		01 FB 0044D	CALLS #1, DBG\$OPCODE_INDEX	
	64		50 D0 00454	MOVL R0, (NOUN_NODE)	
			49 11 00457	BRB 71\$	1226
		00000000'	01 DD 00459 68\$:	PUSHL #1	1287
			EF 9F 0045B	PUSHAB P.ABD	
			53 DD 00461	PUSHL R3	
00000000G	00		03 FB 00463	CALLS #3, DBG\$NMATCH	
	01		50 D1 0046A	CMPL R0, #1	
			08 12 0046D	BNEQ 69\$	
		00000000'	EF 9F 0046F	PUSHAB P.ABE	1288
			1C 11 00475	BRB 70\$	
			03 DD 00477 69\$:	PUSHL #3	1290
		00000000'	EF 9F 00479	PUSHAB P.ABF	
			53 DD 0047F	PUSHL R3	
00000000G	00		03 FB 00481	CALLS #3, DBG\$NMATCH	
	01		50 D1 00488	CMPL R0, #1	
			A5 12 0048B	BNEQ 66\$	
		00000000'	EF 9F 0048D	PUSHAB P.ABG	1291
			01 DD 00493 70\$:	PUSHL #1	
		00028148	8F DD 00495	PUSHL #164168	
00000000G	00		03 FB 0049B	CALLS #3, LIB\$SIGNAL	
		FBFF	31 004A2 71\$:	BRW 10\$	
			01 DD 004A5 72\$:	PUSHL #1	1329
		00000000'	EF 9F 004A7	PUSHAB DBG\$CS_RETURN	
			53 DD 004AD	PUSHL R3	
00000000G	00		03 FB 004AF	CALLS #3, DBG\$NMATCH	
	01		50 D1 004B6	CMPL R0, #1	
			07 12 004B9	BNEQ 73\$	
			57 D4 004BB	CLRL CMD_KIND	1331
	55		04 D0 004BD	MOVL #4, SUB_KIND	1332
			74 11 004C0	BRB 77\$	1323
		00000000'	02 DD 004C2 73\$:	PUSHL #2	1338
			EF 9F 004C4	PUSHAB DBG\$CS_SOURCE	
			53 DD 004CA	PUSHL R3	
00000000G	00		03 FB 004CC	CALLS #3, DBG\$NMATCH	
	01		50 D1 004D3	CMPL R0, #1	
			09 12 004D6	BNEQ 74\$	
14	AE		01 D0 004D8	MOVL #1, SOURCE	1340
		10	AE D4 004DC	CLRL NOSOURCE	1341

		00000000'	72	11	004DF	BRB	79\$		1323
			04	DD	004E1	PUSHL	#4		1347
			EF	9F	004E3	PUSHAB	DBG\$CS_NOSOURCE		
			53	DD	004E9	PUSHL	R3		
00000000G	00		03	FB	004EB	CALLS	#3, DBG\$NMATCH		
	01		50	D1	004F2	CMPL	R0, #1		
			06	12	004F5	BNEQ	75\$		
10	AE		01	7D	004F7	MOVQ	#1, NOSOURCE		1349
			71	11	004FB	BRB	81\$		1323
			02	DD	004FD	PUSHL	#2		1356
		00000000'	EF	9F	004FF	PUSHAB	DBG\$CS_SYSTEM		
			53	DD	00505	PUSHL	R3		
00000000G	00		03	FB	00507	CALLS	#3, DBG\$NMATCH		
	01		50	D1	0050E	CMPL	R0, #1		
			09	12	00511	BNEQ	76\$		
0C	AE		01	D0	00513	MOVL	#1, SYSTEM		1358
		08	AE	D4	00517	CLRL	NOSYSTEM		1359
			70	11	0051A	BRB	83\$		1323
			04	DD	0051C	PUSHL	#4		1365
		00000000'	EF	9F	0051E	PUSHAB	DBG\$CS_NOSYSTEM		
			53	DD	00524	PUSHL	R3		
00000000G	00		03	FB	00526	CALLS	#3, DBG\$NMATCH		
	01		50	D1	0052D	CMPL	R0, #1		
			06	12	00530	BNEQ	78\$		
08	AE		01	7D	00532	MOVQ	#1, NOSYSTEM		1367
			6F	11	00536	BRB	85\$		1323
			02	DD	00538	PUSHL	#2		1374
		00000000'	EF	9F	0053A	PUSHAB	DBG\$CS_SILENT		
			53	DD	00540	PUSHL	R3		
00000000G	00		03	FB	00542	CALLS	#3, DBG\$NMATCH		
	01		50	D1	00549	CMPL	R0, #1		
			07	12	0054C	BNEQ	80\$		
	5B		01	D0	0054E	MOVL	#1, SILENT		1376
			5A	D4	00551	CLRL	NOSILENT		1377
			70	11	00553	BRB	87\$		1323
			04	DD	00555	PUSHL	#4		1383
		00000000'	EF	9F	00557	PUSHAB	DBG\$CS_NOSILENT		
			53	DD	0055D	PUSHL	R3		
00000000G	00		03	FB	0055F	CALLS	#3, DBG\$NMATCH		
	01		50	D1	00566	CMPL	R0, #1		
			05	12	00569	BNEQ	82\$		
	5A		01	7D	0056B	MOVQ	#1, NOSILENT		1386
			73	11	0056E	BRB	89\$		1323
			01	DD	00570	PUSHL	#1		1392
		00000000'	EF	9F	00572	PUSHAB	DBG\$CS_INT0		
			53	DD	00578	PUSHL	R3		
00000000G	00		03	FB	0057A	CALLS	#3, DBG\$NMATCH		
	01		50	D1	00581	CMPL	R0, #1		
			08	12	00584	BNEQ	84\$		
04	AE		01	D0	00586	MOVL	#1, INTO		1394
			6E	D4	0058A	CLRL	OVER		1395
			73	11	0058C	BRB	91\$		1323
			01	DD	0058E	PUSHL	#1		1401
		00000000'	EF	9F	00590	PUSHAB	DBG\$CS_OVER		
			53	DD	00596	PUSHL	R3		
00000000G	00		03	FB	00598	CALLS	#3, DBG\$NMATCH		
	01		50	D1	0059F	CMPL	R0, #1		

		05	12	005A2	BNEQ	86\$		
	6E	01	7D	005A4	MOVQ	#1	OVER	1403
		76	11	005A7	BRB	93\$		1323
		01	DD	005A9	PUSHL	#1		1410
	00000000'	EF	9F	005AB	PUSHAB	DBG\$CS_CALL		
		53	DD	005B1	PUSHL	R3		
00000000G	00	03	FB	005B3	CALLS	#3, DBG\$NMATCH		
	01	50	D1	005BA	CMPL	R0, #1		
		08	12	005BD	BNEQ	88\$		
	57	01	DD	005BF	MOVL	#1, CMD_KIND		1412
	55	05	DD	005C2	MOVL	#5, SUB_KIND		1413
		58	11	005C5	BRB	93\$		1323
		01	DD	005C7	PUSHL	#1		1419
	00000000'	EF	9F	005C9	PUSHAB	DBG\$CS_BRANCH		
		53	DD	005CF	PUSHL	R3		
00000000G	00	03	FB	005D1	CALLS	#3, DBG\$NMATCH		
	01	50	D1	005D8	CMPL	R0, #1		
		08	12	005DB	BNEQ	90\$		
	57	01	DD	005DD	MOVL	#1, CMD_KIND		1421
	55	06	DD	005E0	MOVL	#6, SUB_KIND		1422
		3A	11	005E3	BRB	93\$		1323
		01	DD	005E5	PUSHL	#1		1428
	00000000'	EF	9F	005E7	PUSHAB	DBG\$CS_EXCEPTION		
		53	DD	005ED	PUSHL	R3		
00000000G	00	03	FB	005EF	CALLS	#3, DBG\$NMATCH		
	01	50	D1	005F6	CMPL	R0, #1		
		08	12	005F9	BNEQ	92\$		
	57	02	DD	005FB	MOVL	#2, CMD_KIND		1430
	55	0F	DD	005FE	MOVL	#15, SUB_KIND		1431
		1C	11	00601	BRB	93\$		1323
		01	DD	00603	PUSHL	#1		1437
	00000000'	EF	9F	00605	PUSHAB	DBG\$CS_LINE		
		53	DD	0060B	PUSHL	R3		
00000000G	00	03	FB	0060D	CALLS	#3, DBG\$NMATCH		
	01	50	D1	00614	CMPL	R0, #1		
		09	12	00617	BNEQ	94\$		
	57	01	DD	00619	MOVL	#1, CMD_KIND		1439
	55	07	DD	0061C	MOVL	#7, SUB_KIND		1440
		0104	31	0061F	BRW	102\$		1323
		01	DD	00622	PUSHL	#1		1447
	00000000'	EF	9F	00624	PUSHAB	DBG\$CS_INSTRUCTION		
		53	DD	0062A	PUSHL	R3		
00000000G	00	03	FB	0062C	CALLS	#3, DBG\$NMATCH		
	01	50	D1	00633	CMPL	R0, #1		
		03	13	00636	BEQL	95\$		
		00A2	31	00638	BRW	99\$		
	57	01	DD	0063B	MOVL	#1, CMD_KIND		1449
	55	08	DD	0063E	MOVL	#8, SUB_KIND		1450
		01	DD	00641	PUSHL	#1		1455
	00000000'	EF	9F	00643	PUSHAB	DBG\$CS_EQUALS		
		53	DD	00649	PUSHL	R3		
00000000G	00	03	FB	0064B	CALLS	#3, DBG\$NMATCH		
	CA	50	E9	00652	BLBC	R0, 93\$		
	55	09	DD	00655	MOVL	#9, SUB_KIND		1461
		01	DD	00658	PUSHL	#1		1465
	00000000'	EF	9F	0065A	PUSHAB	DBG\$CS_L_PAREN		
		53	DD	00660	PUSHL	R3		

00000000G	00	03	FB	00662	CALLS	#3, DBG\$NMATCH	
	4D	50	E9	00669	BLBC	R0, 98\$	
		04	DD	0066C	PUSHL	#4	1472
00000000G	00	01	FB	0066E	CALLS	#1, DBG\$GET TEMPMEM	
	54	50	D0	00675	MOVL	R0, NOUN_NODE	
		08	A4	D4 00678	CLRL	8(NOUN_NODE)	
	69	54	D0	0067B	MOVL	NOUN_NODE, (NOUN_LINK)	
	59	08	A4	9E 0067E	MOVAB	8(NOUN_NODE), NOON_LINK	
		53	DD	00682	PUSHL	R3	1479
00000000G	00	01	FB	00684	CALLS	#1, DBG\$OPCODE_INDEX	
	64	50	D0	0068B	MOVL	R0, (NOUN_NODE)	
		01	DD	0068E	PUSHL	#1	1482
	00000000'	EF	9F	00690	PUSHAB	DBG\$CS_COMMA	
		53	DD	00696	PUSHL	R3	
00000000G	00	03	FB	00698	CALLS	#3, DBG\$NMATCH	
	CA	50	E8	0069F	BLBS	R0, 96\$	
		01	DD	006A2	PUSHL	#1	1487
	00000000'	EF	9F	006A4	PUSHAB	DBG\$CS_R_PAREN	
		53	DD	006AA	PUSHL	R3	
00000000G	00	03	FB	006AC	CALLS	#3, DBG\$NMATCH	
	70	50	E8	006B3	BLBS	R0, 102\$	
		037F	31	006B6	BRW	138\$	
		04	DD	006B9	PUSHL	#4	1491
00000000G	00	01	FB	006BB	CALLS	#1, DBG\$GET TEMPMEM	
	54	50	D0	006C2	MOVL	R0, NOUN_NODE	
		08	A4	D4 006C5	CLRL	8(NOUN_NODE)	
	69	54	D0	006C8	MOVL	NOUN_NODE, (NOUN_LINK)	
	59	08	A4	9E 006CB	MOVAB	8(NOUN_NODE), NOON_LINK	
		53	DD	006CF	PUSHL	R3	1497
00000000G	00	01	FB	006D1	CALLS	#1, DBG\$OPCODE_INDEX	
	64	50	D0	006D8	MOVL	R0, (NOUN_NODE)	
		49	11	006DB	BRB	102\$	1455
		01	DD	006DD	PUSHL	#1	1507
	00000000'	EF	9F	006DF	PUSHAB	P.ABH	
		53	DD	006E5	PUSHL	R3	
00000000G	00	03	FB	006E7	CALLS	#3, DBG\$NMATCH	
	01	50	D1	006EE	CML	R0, #1	
		08	12	006F1	BNEQ	100\$	
	00000000'	EF	9F	006F3	PUSHAB	P.ABI	1508
		1C	11	006F9	BRB	101\$	
		03	DD	006FB	PUSHL	#3	1510
	00000000'	EF	9F	006FD	PUSHAB	P.ABJ	
		53	DD	00703	PUSHL	R3	
00000000G	00	03	FB	00705	CALLS	#3, DBG\$NMATCH	
	01	50	D1	0070C	CML	R0, #1	
		A5	12	0070F	BNEQ	97\$	
	00000000'	EF	9F	00711	PUSHAB	P.ABK	1511
		01	DD	00717	PUSHL	#1	
	00028148	8F	DD	00719	PUSHL	#164168	
00000000G	00	03	FB	0071F	CALLS	#3, LIB\$SIGNAL	
		01	DD	00726	PUSHL	#1	1523
	00000000'	EF	9F	00728	PUSHAB	DBG\$CS_COMMA	
		53	DD	0072E	PUSHL	R3	
00000000G	00	03	FB	00730	CALLS	#3, DBG\$NMATCH	
	03	50	E9	00737	BLBC	R0, 103\$	
		FD68	31	0073A	BRW	72\$	
		03	DD	0073D	PUSHL	#3	1525

00000000G	00	01	FB	0073F	CALLS	#1, DBG\$GET TEMPMEM	:	
	52	50	D0	00746	MOVL	R0, ADVERB_NODE	:	
		08	A2	D4	00749	CLRL	8(ADVERB_NODE)	:
	68	52	D0	0074C	MOVL	ADVERB_NODE, (ADVERB_LINK)	:	
	58	08	A2	9E	0074F	MOVAB	8(ADVERB_NODE), ADVERB_LINK	:
		57	D5	00753	TSTL	CMD_KIND	1538	
		31	12	00755	BNEQ	108\$:	
		55	D5	00757	TSTL	SUB_KIND	1542	
		05	12	00759	BNEQ	104\$:	
	50	06	D0	0075B	MOVL	#6, R0	:	
		6C	11	0075E	BRB	116\$:	
	01	55	D1	00760	104\$:	CMPL SUB_KIND, #1	1544	
		05	12	00763	BNEQ	105\$:	
	50	07	D0	00765	MOVL	#7, R0	:	
		62	11	00768	BRB	116\$:	
	02	55	D1	0076A	105\$:	CMPL SUB_KIND, #2	1546	
		05	12	0076D	BNEQ	106\$:	
	50	08	D0	0076F	MOVL	#8, R0	:	
		58	11	00772	BRB	116\$:	
	03	55	D1	00774	106\$:	CMPL SUB_KIND, #3	1548	
		05	12	00777	BNEQ	107\$:	
	50	09	D0	00779	MOVL	#9, R0	:	
		4E	11	0077C	BRB	116\$:	
	04	55	D1	0077E	107\$:	CMPL SUB_KIND, #4	1550	
		41	12	00781	BNEQ	114\$:	
	50	0A	D0	00783	MOVL	#10, R0	:	
		44	11	00786	BRB	116\$	1540	
	02	57	D1	00788	108\$:	CMPL CMD_KIND, #2	1556	
		05	12	0078B	BNEQ	109\$:	
	62	0B	90	0078D	MOVB	#11, (ADVERB_NODE)	1557	
		3D	11	00790	BRB	117\$:	
	01	57	D1	00792	109\$:	CMPL CMD_KIND, #1	1560	
		38	12	00795	BNEQ	117\$:	
	05	55	D1	00797	CMPL	SUB_KIND, #5	1564	
		05	12	0079A	BNEQ	110\$:	
	50	12	D0	0079C	MOVL	#18, R0	:	
		2B	11	0079F	BRB	116\$:	
	06	55	D1	007A1	110\$:	CMPL SUB_KIND, #6	1566	
		05	12	007A4	BNEQ	111\$:	
	50	13	D0	007A6	MOVL	#19, R0	:	
		21	11	007A9	BRB	116\$:	
	08	55	D1	007AB	111\$:	CMPL SUB_KIND, #8	1568	
		05	12	007AE	BNEQ	112\$:	
	50	15	D0	007B0	MOVL	#21, R0	:	
		17	11	007B3	BRB	116\$:	
	09	55	D1	007B5	112\$:	CMPL SUB_KIND, #9	1570	
		05	12	007B8	BNEQ	113\$:	
	50	16	D0	007BA	MOVL	#22, R0	:	
		0D	11	007BD	BRB	116\$:	
	07	55	D1	007BF	113\$:	CMPL SUB_KIND, #7	1572	
		05	13	007C2	BEQL	115\$:	
	50	01	CE	007C4	114\$:	MNEGL #1, R0	:	
		03	11	007C7	BRB	116\$:	
	50	14	D0	007C9	115\$:	MOVL #20, R0	:	
	62	50	90	007CC	116\$:	MOVB R0, (ADVERB_NODE)	1562	
	01	20	AE	D1	007CF	117\$:	CMPL AFTER_COUNT, #1	1583
		27	13	007D3	BEQL	119\$:	

00000000G	00	03	DD	007D5	PUSHL	#3	1585	
	52	01	FB	007D7	CALLS	#1, DBG\$GET TEMPMEM		
		50	D0	007DE	MOVL	R0, ADVERB NODE		
	68	08	A2	D4 007E1	CLRL	8(ADVERB NODE)		
	58		52	D0 007E4	MOVL	ADVERB NODE, (ADVERB LINK)		
		08	A2	9E 007E7	MOVAB	8(ADVERB NODE), ADVERB_LINK		
		20	AE	D5 007EB	TSTL	AFTER_COUNT	1592	
	62		05	12 007EE	BNEQ	118\$		
			01	90 007F0	MOVB	#1, (ADVERB_NODE)	1594	
			07	11 007F3	BRB	119\$		
			62	94 007F5	118\$:	CLRB	(ADVERB NODE)	1597
04	A2	20	AE	D0 007F7	MOVL	AFTER_COUNT, 4(ADVERB_NODE)	1598	
	1B		5B	E9 007FC	119\$:	BLBC	SILENT, 120\$	1606
			03	DD 007FF	PUSHL	#3	1608	
00000000G	00	01	FB	00801	CALLS	#1, DBG\$GET TEMPMEM		
	52	50	D0	00808	MOVL	R0, ADVERB NODE		
		08	A2	D4 0080B	CLRL	8(ADVERB NODE)		
	68		52	D0 0080E	MOVL	ADVERB NODE, (ADVERB LINK)		
	58	08	A2	9E 00811	MOVAB	8(ADVERB NODE), ADVERB_LINK		
	62		02	90 00815	MOVB	#2, (ADVERB_NODE)	1610	
			1C	11 00818	BRB	121\$	1606	
	19		5A	E9 0081A	120\$:	BLBC	NOSILENT, 121\$	1613
			03	DD 0081D	PUSHL	#3	1615	
00000000G	00	01	FB	0081F	CALLS	#1, DBG\$GET TEMPMEM		
	52	50	D0	00826	MOVL	R0, ADVERB NODE		
		08	A2	D4 00829	CLRL	8(ADVERB NODE)		
	68		52	D0 0082C	MOVL	ADVERB NODE, (ADVERB LINK)		
	58	08	A2	9E 0082F	MOVAB	8(ADVERB NODE), ADVERB_LINK		
	62		03	90 00833	MOVB	#3, (ADVERB_NODE)	1617	
	19	18	AE	E9 00836	121\$:	BLBC	TEMPORARY, T22\$	1623
			03	DD 0083A	PUSHL	#3	1625	
00000000G	00	01	FB	0083C	CALLS	#1, DBG\$GET TEMPMEM		
	52	50	D0	00843	MOVL	R0, ADVERB NODE		
		08	A2	D4 00846	CLRL	8(ADVERB NODE)		
	68		52	D0 00849	MOVL	ADVERB NODE, (ADVERB LINK)		
	58	08	A2	9E 0084C	MOVAB	8(ADVERB NODE), ADVERB_LINK		
	62		01	90 00850	MOVB	#1, (ADVERB_NODE)	1627	
	19	14	AE	E9 00853	122\$:	BLBC	SOURCE, 123\$	1634
			03	DD 00857	PUSHL	#3	1636	
00000000G	00	01	FB	00859	CALLS	#1, DBG\$GET TEMPMEM		
	52	50	D0	00860	MOVL	R0, ADVERB NODE		
		08	A2	D4 00863	CLRL	8(ADVERB NODE)		
	68		52	D0 00866	MOVL	ADVERB NODE, (ADVERB LINK)		
	58	08	A2	9E 00869	MOVAB	8(ADVERB NODE), ADVERB_LINK		
	62		0C	90 0086D	MOVB	#12, (ADVERB_NODE)	1638	
	19	10	AE	E9 00870	123\$:	BLBC	NOSOURCE, 124\$	1641
			03	DD 00874	PUSHL	#3	1643	
00000000G	00	01	FB	00876	CALLS	#1, DBG\$GET TEMPMEM		
	52	50	D0	0087D	MOVL	R0, ADVERB NODE		
		08	A2	D4 00880	CLRL	8(ADVERB NODE)		
	68		52	D0 00883	MOVL	ADVERB NODE, (ADVERB LINK)		
	58	08	A2	9E 00886	MOVAB	8(ADVERB NODE), ADVERB_LINK		
	62		0D	90 0088A	MOVB	#13, (ADVERB_NODE)	1645	
	01		57	D1 0088D	124\$:	CPL	CMD_KIND, #1	1651
			09	13 00890	BEQL	125\$		
			57	D5 00892	TSTL	CMD_KIND	1652	
			78	12 00894	BNEQ	129\$		

04		55	D1	00896	CMPL	SUB KIND, #4	1653
19	0C	73	12	00899	BNEQ	129\$	1656
00000000G	00	AE	E9	0089B	BLBC	SYSTEM, 126\$	1658
52		03	DD	0089F	PUSHL	#3	
68	08	01	FB	008A1	CALLS	#1, DBG\$GET_TEMPMEM	
58		50	D0	008A8	MOVL	R0, ADVERB_NODE	
62	08	A2	D4	008AB	CLRL	8(ADVERB_NODE)	
19		52	D0	008AE	MOVL	ADVERB_NODE, (ADVERB_LINK)	
	08	A2	9E	008B1	MOVAB	8(ADVERB_NODE), ADVERB_LINK	
		0E	90	008B5	MOVB	#14, (ADVERB_NODE)	1660
	08	AE	E9	008B8	BLBC	NOSYSTEM, 127\$	1663
00000000G	00	03	DD	008BC	PUSHL	#3	1665
52		01	FB	008BE	CALLS	#1, DBG\$GET_TEMPMEM	
68	08	50	D0	008C5	MOVL	R0, ADVERB_NODE	
58		A2	D4	008C8	CLRL	8(ADVERB_NODE)	
62	08	52	D0	008CB	MOVL	ADVERB_NODE, (ADVERB_LINK)	
19		A2	9E	008CE	MOVAB	8(ADVERB_NODE), ADVERB_LINK	
	04	0F	90	008D2	MOVB	#15, (ADVERB_NODE)	1667
		AE	E9	008D5	BLBC	INT0, 128\$	1670
00000000G	00	03	DD	008D9	PUSHL	#3	1672
52		01	FB	008DB	CALLS	#1, DBG\$GET_TEMPMEM	
68	08	50	D0	008E2	MOVL	R0, ADVERB_NODE	
58		A2	D4	008E5	CLRL	8(ADVERB_NODE)	
62	08	52	D0	008E8	MOVL	ADVERB_NODE, (ADVERB_LINK)	
19		A2	9E	008EB	MOVAB	8(ADVERB_NODE), ADVERB_LINK	
		10	90	008EF	MOVB	#16, (ADVERB_NODE)	1674
		6E	E9	008F2	BLBC	OVER, 129\$	1677
00000000G	00	03	DD	008F5	PUSHL	#3	1679
52		01	FB	008F7	CALLS	#1, DBG\$GET_TEMPMEM	
68	08	50	D0	008FE	MOVL	R0, ADVERB_NODE	
58		A2	D4	00901	CLRL	8(ADVERB_NODE)	
62	08	52	D0	00904	MOVL	ADVERB_NODE, (ADVERB_LINK)	
03		A2	9E	00907	MOVAB	8(ADVERB_NODE), ADVERB_LINK	
		11	90	0090B	MOVB	#17, (ADVERB_NODE)	1681
		56	D1	0090E	CMPL	CMD_TYPE, #3	1690
		60	13	00911	BEQL	132\$	
		57	D5	00913	TSTL	CMD_KIND	1691
		5C	12	00915	BNEQ	132\$	
		04	DD	00917	PUSHL	#4	1695
00000000G	00	01	FB	00919	CALLS	#1, DBG\$GET_TEMPMEM	
54		50	D0	00920	MOVL	R0, NOUN_NODE	
69	08	A4	D4	00923	CLRL	8(NOUN_NODE)	
59		54	D0	00926	MOVL	NOUN_NODE, (NOUN_LINK)	
00000000'	EF	A4	9E	00929	MOVAB	8(NOUN_NODE), NOUN_LINK	
		01	90	0092D	MOVB	#1, DBG\$GB_SET_BREAK_FLAG	1709
	0C	AC	DD	00934	PUSHL	MESSAGE_VECT	1713
		09	DD	00937	PUSHL	#9	1711
7E 00000000G	00	9A	00939	MOVZBL	DBG\$GB_RADIX, -(SP)		1712
		18	BB	00940	PUSHR	#^M<R3,R4>	1711
00000000G	00	05	FB	00942	CALLS	#5, DBG\$NPARSE_ADDRESS	
55		50	D0	00949	MOVL	R0, STATUS	
	00000000'	EF	94	0094C	CLRB	DBG\$GB_SET_BREAK_FLAG	1714
01		55	D1	00952	CMPL	STATUS, #1	1720
		1C	13	00955	BEQL	132\$	
		55	D5	00957	TSTL	STATUS	1728
		04	13	00959	BEQL	131\$	
50		55	D0	0095B	MOVL	STATUS, R0	1730

			04 0095E	RET		
		01 DD 0095F	131\$:	PUSHL	#1	1733
		EF 9F 00961		PUSHAB	DBG\$CS_COMMA	
		53 DD 00967		PUSHL	R3	
00000000G	00	03 FB 00969		CALLS	#3, DBG\$NMATCH	
	A4	50 E8 00970		BLBS	R0, 130\$	
		01 DD 00973	132\$:	PUSHL	#1	1742
		EF 9F 00975		PUSHAB	DBG\$CS_CR	
		53 DD 0097B		PUSHL	R3	
00000000G	00	03 FB 0097D		CALLS	#3, DBG\$NMATCH	
	33	50 E8 00984		BLBS	R0, 134\$	
	03	56 D1 00987		CMPL	CMD TYPE, #3	1749
		37 12 0098A		BNEQ	136\$	
	03	AE E8 0098C		BLBS	28(SP), 133\$	1770
		00A5 31 00990		BRW	138\$	
		03 DD 00993	133\$:	PUSHL	#3	1753
00000000G	00	01 FB 00995		CALLS	#1, DBG\$GET_TEMP MEM	
	52	50 D0 0099C		MOVL	R0, ADVERB_NODE	
		08 A2 D4 0099F		CLRL	8(ADVERB_NODE)	
	68	52 D0 009A2		MOVL	ADVERB_NODE, (ADVERB_LINK)	
	58	08 A2 9E 009A5		MOVAB	8(ADVERB_NODE), ADVERB_LINK	
		62 94 009A9		CLRB	(ADVERB_NODE)	1766
		0C AC DD 009AB		PUSHL	MESSAGE_VECT	1770
		04 A2 9F 009AE		PUSHAB	4(ADVERB_NODE)	1769
		53 DD 009B1		PUSHL	R3	
00000000G	00	03 FB 009B3		CALLS	#3, DBG\$NSAVE_DECIMAL_INTEGER	
	03	50 E9 009BA	134\$:	BLBC	R0, 135\$	
		00D6 31 009BD		BRW	142\$	
		00ED 31 009C0	135\$:	BRW	145\$	1775
		01 DD 009C3	136\$:	PUSHL	#1	1786
		EF 9F 009C5		PUSHAB	DBG\$CS_WHEN	
		53 DD 009CB		PUSHL	R3	
00000000G	00	03 FB 009CD		CALLS	#3, DBG\$NMATCH	
	39	50 E9 009D4		BLBC	R0, 137\$	
		01 DD 009D7		PUSHL	#1	1793
		EF 9F 009D9		PUSHAB	DBG\$CS_L_PAREN	
		53 DD 009DF		PUSHL	R3	
00000000G	00	03 FB 009E1		CALLS	#3, DBG\$NMATCH	
	4D	50 E9 009E8		BLBC	R0, 138\$	
		03 DD 009EB		PUSHL	#3	
00000000G	00	01 FB 009ED		CALLS	#1, DBG\$GET_TEMP MEM	
	52	50 D0 009F4		MOVL	R0, ADVERB_NODE	
		08 A2 D4 009F7		CLRL	8(ADVERB_NODE)	
	68	52 D0 009FA		MOVL	ADVERB_NODE, (ADVERB_LINK)	
	58	08 A2 9E 009FD		MOVAB	8(ADVERB_NODE), ADVERB_LINK	
	62	04 90 00A01		MOVB	#4, (ADVERB_NODE)	1803
		A2 9F 00A04		PUSHAB	4(ADVERB_NODE)	1804
		53 DD 00A07		PUSHL	R3	
00000000G	00	02 FB 00A09		CALLS	#2, DBG\$NSAVE_BREAK_BUFFER	
		01 DD 00A10	137\$:	PUSHL	#1	1810
		EF 9F 00A12		PUSHAB	DBG\$CS_DO	
		53 DD 00A18		PUSHL	R3	
00000000G	00	03 FB 00A1A		CALLS	#3, DBG\$NMATCH	
	5E	50 E9 00A21		BLBC	R0, 141\$	
		01 DD 00A24		PUSHL	#1	1817
		EF 9F 00A26		PUSHAB	DBG\$CS_L_PAREN	
		53 DD 00A2C		PUSHL	R3	

00000000G	00	03	FB	00A2E	CALLS	#3, DBG\$NMATCH	
	23	50	E8	00A35	BLBS	R0, 140\$	
		01	DD	00A38	138\$: PUSHL	#1	
		EF	9F	00A3A	PUSHAB	DBG\$CS_CR	
	00000000'	53	DD	00A40	PUSHL	R3	
00000000G	00	03	FB	00A42	CALLS	#3, DBG\$NMATCH	
	4E	50	E9	00A49	BLBC	R0, 143\$	
		8F	DD	00A4C	139\$: PUSHL	#164048	
00000000G	00	01	FB	00A52	CALLS	#1, DBG\$NMAKE_ARG_VECT	
		51	11	00A59	BRB	144\$	
		03	DD	00A5B	140\$: PUSHL	#3	
00000000G	00	01	FB	00A5D	CALLS	#1, DBG\$GET_TEMP_MEM	
	52	50	D0	00A64	MOVL	R0, ADVERB_NODE	
		A2	D4	00A67	CLRL	8(ADVERB_NODE)	
	68	52	D0	00A6A	MOVL	ADVERB_NODE, (ADVERB_LINK)	
	58	A2	9E	00A6D	MOVAB	8(ADVERB_NODE), ADVERB_LINK	
	62	05	90	00A71	MOVB	#5, (ADVERB_NODE)	1827
		A2	9F	00A74	PUSHAB	4(ADVERB_NODE)	1828
		53	DD	00A77	PUSHL	R3	
00000000G	00	02	FB	00A79	CALLS	#2, DBG\$NSAVE_BREAK_BUFFER	
		14	11	00A80	BRB	142\$	1829
		01	DD	00A82	141\$: PUSHL	#1	1836
		EF	9F	00A84	PUSHAB	DBG\$CS_CR	
	00000000'	53	DD	00A8A	PUSHL	R3	
00000000G	00	03	FB	00A8C	CALLS	#3, DBG\$NMATCH	
	04	50	E9	00A93	BLBC	R0, 143\$	
	50	01	D0	00A96	142\$: MOVL	#1, R0	
			04	00A99	RET		
		53	DD	00A9A	143\$: PUSHL	R3	1837
00000000G	00	01	FB	00A9C	CALLS	#1, DBG\$NNEXT_WORD	
		50	DD	00AA3	PUSHL	R0	
00000000G	00	01	FB	00AA5	CALLS	#1, DBG\$NSYNTAX_ERROR	
	0C	50	D0	00AAC	144\$: MOVL	R0, @MESSAGE_VECT	
		04	D0	00AB0	145\$: MOVL	#4, R0	1838
			04	00AB3	RET		1839

; Routine Size: 2740 bytes, Routine Base: DBG\$CODE + 0037

```
: 1715      1840 1 GLOBAL ROUTINE DBG$EVENT_SEMANTICS(VERB_NODE, MESSAGE_VECT) =
: 1716      1841 1
: 1717      1842 1 FUNCTION
: 1718      1843 1     This routine builds the event data structures based on the
: 1719      1844 1     parse tree passed via the VERB_NODE address. This routine
: 1720      1845 1     is called from DBG$NSET.
: 1721      1846 1
: 1722      1847 1 INPUTS
: 1723      1848 1     VERB_NODE:      A pointer to a DEBUG verb node, which defines,
: 1724      1849 1     along with the verb's descendants, the desired
: 1725      1850 1     command.
: 1726      1851 1     MESSAGE_VECT:    The address of a longword to contain the address
: 1727      1852 1     of a message argument vector.
: 1728      1853 1
: 1729      1854 1 OUTPUTS
: 1730      1855 1     The event data structure(s) are updated to include the new event,
: 1731      1856 1     and may include:
: 1732      1857 1
: 1733      1858 1     EVENT$CMD QUEUE,
: 1734      1859 1     EVENT$WHEN QUEUE,
: 1735      1860 1     EVENT$DO LIST QUEUE,
: 1736      1861 1     EVENT$PAGE_QUEUE
: 1737      1862 1
: 1738      1863 1     ST$K_SEVERE is return on error, otherwise ST$K_SUCCESS
: 1739      1864 1
: 1740      1865 2 BEGIN
: 1741      1866 2
: 1742      1867 2 MAP
: 1743      1868 2     VERB_NODE :      REF DBG$VERB_NODE;      ! Verb node pointer
: 1744      1869 2
: 1745      1870 2 LOCAL
: 1746      1871 2     DUMMY,
: 1747      1872 2     EVENT_QUEUE :    QUEUE HEAD,      ! Working queue head
: 1748      1873 2     QUEUE_ENTRY :    REF EVENT$EVENT_DESCRIPTOR, ! Event queue entry pointer
: 1749      1874 2     NEXTQ_ENTRY :    REF EVENT$EVENT_DESCRIPTOR, ! Event queue entry pointer
: 1750      1875 2     EVENT_ENTRY :    REF EVENT$EVENT_DESCRIPTOR, ! Event entry pointer
: 1751      1876 2     DO_ENTRY :      REF EVENT$DO LIST_DESCRIPTOR, ! DO List entry pointer
: 1752      1877 2     WHEN_ENTRY :    REF EVENT$WHEN_DESCRIPTOR, ! WHEN entry pointer
: 1753      1878 2     ADVERB_NODE :    REF DBG$ADVERB_NODE,      ! Adverb node link
: 1754      1879 2     NOUN_NODE :      REF DBG$NOUN_NODE,        ! Noun node link
: 1755      1880 2
: 1756      1881 2     PRIMARY :      REF DBG$PRIMARY,      ! Event primary descriptor
: 1757      1882 2     VALUE :        REF DBG$VALDESC,      ! Event value descriptor
: 1758      1883 2
: 1759      1884 2     ADDRESS :      VECTOR [2],      ! Event address
: 1760      1885 2     ADR_TYP,      ! Event address type
: 1761      1886 2     LENGTH,      ! Event value length
: 1762      1887 2
: 1763      1888 2     STRING :        REF VECTOR [, BYTE],    ! Text pointer
: 1764      1889 2     COUNTER;      ! Counter variable
: 1765      1890 2
: 1766      1891 2
: 1767      1892 2     ! Initialize the working queue to be empty.
: 1768      1893 2
: 1769      1894 2     EVENT_QUEUE [L_QUEUE_FLINK] = EVENT_QUEUE;
: 1770      1895 2     EVENT_QUEUE [L_QUEUE_BLINK] = EVENT_QUEUE;
: 1771      1896 2
```



```
: 1772      1897      2
: 1773      1898      2
: 1774      1899      2
: 1775      1900      2
: 1776      1901      2
: 1777      1902      2
: 1778      1903      2
: 1779      1904      2
: 1780      1905      2
: 1781      1906      2
: 1782      1907      2
: 1783      1908      2
: 1784      1909      2
: 1785      1910      2
: 1786      1911      2
: 1787      1912      2
: 1788      1913      2
: 1789      1914      2
: 1790      1915      2
: 1791      1916      2
: 1792      1917      2
: 1793      1918      2
: 1794      1919      2
: 1795      1920      2
: 1796      1921      2
: 1797      1922      2
: 1798      1923      2
: 1799      1924      2
: 1800      1925      2
: 1801      1926      2
: 1802      1927      2
: 1803      1928      2
: 1804      1929      2
: 1805      1930      2
: 1806      1931      2
: 1807      1932      2
: 1808      1933      2
: 1809      1934      2
: 1810      1935      2
: 1811      1936      2
: 1812      1937      2
: 1813      1938      2
: 1814      1939      2
: 1815      1940      2
: 1816      1941      2
: 1817      1942      2
: 1818      1943      2
: 1819      1944      2
: 1820      1945      2
: 1821      1946      2
: 1822      1947      2
: 1823      1948      2
: 1824      1949      2
: 1825      1950      2
: 1826      1951      2
: 1827      1952      2
: 1828      1953      2

! Get an event descriptor from permanent memory, and
! link it into the working queue. Also initialize the
! inside queue header.
EVENT_ENTRY = DBGSGET_MEMORY (EVENTSK_EVENT_DESCRIPTOR_SIZE);
INSQUE (.EVENT_ENTRY, EVENT_QUEUE);
EVENT_ENTRY[EVENTSL_EXC_FLINK] = .EVENT_ENTRY;
EVENT_ENTRY[EVENTSL_EXC_BLINK] = .EVENT_ENTRY;

! Set the event's command type, found in the verb.
SELECTONE .VERB_NODE [DBG$B_VERB_COMPOSITE] OF
SET
  [EVENTSK_SET_BREAK]:
    EVENT_ENTRY [EVENTSB_CMD_TYPE] = EVENTSK_TYPE_BREAK;

  [EVENTSK_SET_BREAK_EXC]:
    BEGIN
      EVENT_ENTRY [EVENTSB_CMD_TYPE] = EVENTSK_TYPE_BREAK;
      EVENT_ENTRY [EVENTSB_CMD_KIND] = EVENTSK_KIND_EXC;
    END;

  [EVENTSK_SET_TRACE]:
    EVENT_ENTRY [EVENTSB_CMD_TYPE] = EVENTSK_TYPE_TRACE;

  [EVENTSK_SET_WATCH]:
    BEGIN
      EVENT_ENTRY [EVENTSB_CMD_TYPE] = EVENTSK_TYPE_WATCH;
      EVENT_ENTRY [EVENTSB_CMD_KIND] = EVENTSK_KIND_ACC;
      EVENT_ENTRY [EVENTSB_SUB_KIND] = EVENTSK_ACC_MDFY;
    END;

  [EVENTSK_SET_STEP,
  EVENTSK_STEP]:
    EVENT_ENTRY [EVENTSB_CMD_TYPE] = EVENTSK_TYPE_STEPS;

  [OTHERWISE]:
    $DBG_ERROR('DBGEVENT\EVENT_SEMANTICS 10');

TES;

! Initialize the Deleted, Silent, Once (Temporary), Threaded, and
! Has Value Descriptor flags.
EVENT_ENTRY [EVENTSB_CMD_FLAGS] = 0;

! Initialize the step-type flags. If the command is STEP or SET STEP,
! we set up the Event Entry using the current default stepping modes.
EVENT_ENTRY[EVENTSL_STEP_FLAGS] = 0;
IF .EVENT_ENTRY [EVENTSB_CMD_TYPE] EQLU EVENTSK_TYPE_STEPS
THEN
```

```

: 1829      1954      3      BEGIN
: 1830      1955      3      EVENT_ENTRY[EVENT$V_STEP_LINE] = .DBG$GB_STP_PTR[EVENT$V_STEPPING_LINE];
: 1831      1956      3      EVENT_ENTRY[EVENT$V_STEP_OVER] = .DBG$GB_STP_PTR[EVENT$V_STEPPING_OVER];
: 1832      1957      3      EVENT_ENTRY[EVENT$V_STEP_SOURCE] =
: 1833      1958      3      .DBG$GB_STP_PTR[EVENT$V_STEPPING_SOURCE];
: 1834      1959      3      EVENT_ENTRY[EVENT$V_OVRD_SOURCE] = FALSE;
: 1835      1960      3      EVENT_ENTRY[EVENT$V_STEP_NOSYSTEM] =
: 1836      1961      3      .DBG$GB_STP_PTR [EVENT$V_STEPPING_NOSYSTEM];
: 1837      1962      3      EVENT_ENTRY[EVENT$V_STEP NOSILENT] =
: 1838      1963      3      .DBG$GB_STP_PTR [EVENT$V_STEPPING_NOSILENT];
: 1839      1964      3      END
: 1840      1965      3
: 1841      1966      3
: 1842      1967      3      ! If the command is SET BREAK, SET TRACE, or SET WATCH, we initialize the
: 1843      1968      3      ! various flags to FALSE.
: 1844      1969      3
: 1845      1970      2      ELSE
: 1846      1971      2      BEGIN
: 1847      1972      2      EVENT_ENTRY[EVENT$V_STEP_LINE] = FALSE;
: 1848      1973      2      EVENT_ENTRY[EVENT$V_STEP_OVER] = FALSE;
: 1849      1974      2      EVENT_ENTRY[EVENT$V_STEP_SOURCE] = FALSE;
: 1850      1975      2      EVENT_ENTRY[EVENT$V_OVRD_SOURCE] = FALSE;
: 1851      1976      2      EVENT_ENTRY[EVENT$V_STEP_NOSYSTEM] = FALSE;
: 1852      1977      2      EVENT_ENTRY[EVENT$V_STEP_NOSILENT] = FALSE;
: 1853      1978      2      END;
: 1854      1979      2
: 1855      1980      2
: 1856      1981      2      ! Initialize the Primary pointer to NULL.
: 1857      1982      2
: 1858      1983      2      EVENT_ENTRY [EVENT$L_PRIMARY] = 0;
: 1859      1984      2
: 1860      1985      2
: 1861      1986      2      ! Initialize the event entry /AFTER count to 1.
: 1862      1987      2
: 1863      1988      2      EVENT_ENTRY [EVENT$L_AFTER_COUNT] = 1;
: 1864      1989      2
: 1865      1990      2
: 1866      1991      2      ! Initialize the WHEN and DO List pointers to NULL.
: 1867      1992      2
: 1868      1993      2      EVENT_ENTRY [EVENT$L_WHEN] = 0;
: 1869      1994      2      EVENT_ENTRY [EVENT$L_DO] = 0;
: 1870      1995      2
: 1871      1996      2
: 1872      1997      2      ! Point to the verb's noun list.
: 1873      1998      2
: 1874      1999      2      NOUN_NODE = .VERB_NODE [DBG$L_VERB_OBJECT_PTR];
: 1875      2000      2
: 1876      2001      2
: 1877      2002      2      ! Point to the verb's adverb list.
: 1878      2003      2
: 1879      2004      2      ADVERB_NODE = .VERB_NODE [DBG$L_VERB_ADVERB_PTR];
: 1880      2005      2
: 1881      2006      2
: 1882      2007      2      ! For each existing adverb, process it.
: 1883      2008      2
: 1884      2009      2      WHILE .ADVERB_NODE NEQA 0 DO
: 1885      2010      2      BEGIN
```


1886 2011
1887 2012
1888 2013
1889 2014
1890 2015
1891 2016
1892 2017
1893 2018
1894 2019
1895 2020
1896 2021
1897 2022
1898 2023
1899 2024
1900 2025
1901 2026
1902 2027
1903 2028
1904 2029
1905 2030
1906 2031
1907 2032
1908 2033
1909 2034
1910 2035
1911 2036
1912 2037
1913 2038
1914 2039
1915 2040
1916 2041
1917 2042
1918 2043
1919 2044
1920 2045
1921 2046
1922 2047
1923 2048
1924 2049
1925 2050
1926 2051
1927 2052
1928 2053
1929 2054
1930 2055
1931 2056
1932 2057
1933 2058
1934 2059
1935 2060
1936 2061
1937 2062
1938 2063
1939 2064
1940 2065
1941 2066
1942 2067

```
! Process adverbs based on the literal set by the
! syntactic processing.
SELECTONE .ADVERB_NODE [DBG$B_ADVERB_LITERAL] OF
SET

! Process the /AFTER : <n> adverb.
[ADVERB AFTER]:
    EVENT_ENTRY [EVENT$A_AFTER_COUNT] =
        .ADVERB_NODE [DBG$L_ADVERB_VALUE];

! Process the /TEMPORARY adverb.
[ADVERB TEMPORARY]:
    EVENT_ENTRY [EVENT$V_ONCE_ONLY] = TRUE;

! Process the /SILENT adverb.
[ADVERB SILENT]:
    BEGIN
    EVENT_ENTRY [EVENT$V_SILENT] = TRUE;
    EVENT_ENTRY [EVENT$V_STEP_NOSILENT] = FALSE;
    EVENT_ENTRY [EVENT$V_OVRD_NOSILENT] = TRUE;
    END;

! Process the /NOSILENT adverb.
[ADVERB NOSILENT]:
    BEGIN
    EVENT_ENTRY [EVENT$V_SILENT] = FALSE;
    EVENT_ENTRY [EVENT$V_STEP_NOSILENT] = TRUE;
    EVENT_ENTRY [EVENT$V_OVRD_NOSILENT] = TRUE;
    END;

! Process the WHEN adverb.
[ADVERB WHEN]:
    BEGIN
    LOCAL
        LENGTH;

    ! Get a WHEN descriptor from permanent memory, and
    ! link it into the WHEN queue.
    WHEN_ENTRY = DBG$GET_MEMORY (EVENT$K_WHEN_DESCRIPTOR_SIZE);
    INSQDE (.WHEN_ENTRY, EVENT$WHEN_QUEUE);

    ! Point the event entry to the WHEN entry.
```



```
: 1943      2068      4      !
: 1944      2069      4      ! EVENT_ENTRY [EVENT$L_WHEN] = .WHEN_ENTRY;
: 1945      2070      4
: 1946      2071      4
: 1947      2072      4      ! Put a <cr> at the end of the WHEN string.
: 1948      2073      4      !
: 1949      2074      4      STRING = .ADVERB_NODE [DBG$L_ADVERB_VALUE];
: 1950      2075      4      LENGTH = .(.STRING)<0,16,0>;
: 1951      2076      4      STRING [.LENGTH+1] = DBG$K_CAR_RETURN;
: 1952      2077      4
: 1953      2078      4
: 1954      2079      4      ! Fill in the WHEN entry.
: 1955      2080      4      !
: 1956      2081      4      WHEN_ENTRY [EVENT$L_WHEN_COUNT] = 1;
: 1957      2082      4      WHEN_ENTRY [EVENT$L_WHEN_POINT] = .STRING;
: 1958      2083      4      END;
: 1959      2084      4
: 1960      2085      4
: 1961      2086      4      ! Process the DO adverb.
: 1962      2087      4      !
: 1963      2088      4      [ADVERB_DO]:
: 1964      2089      4      BEGIN
: 1965      2090      4      LOCAL
: 1966      2091      4      LENGTH;
: 1967      2092      4
: 1968      2093      4
: 1969      2094      4      ! Get a DO LIST descriptor from permanent memory, and
: 1970      2095      4      ! link it into the DO LIST queue.
: 1971      2096      4      !
: 1972      2097      4      DO_ENTRY = DBG$GET_MEMORY (EVENT$K_DO_LIST_DESCRIPTOR_SIZE);
: 1973      2098      4      INSQUE (.DO_ENTRY, EVENT$DO_LIST_QUEUE);
: 1974      2099      4
: 1975      2100      4
: 1976      2101      4      ! Point the event entry to the DO LIST entry.
: 1977      2102      4      !
: 1978      2103      4      EVENT_ENTRY [EVENT$L_DO] = .DO_ENTRY;
: 1979      2104      4
: 1980      2105      4
: 1981      2106      4      ! Put a <cr> at the end of the DO string.
: 1982      2107      4      !
: 1983      2108      4      STRING = .ADVERB_NODE [DBG$L_ADVERB_VALUE];
: 1984      2109      4      LENGTH = .(.STRING)<0,16,0>;
: 1985      2110      4      STRING [.LENGTH+1] = DBG$K_CAR_RETURN;
: 1986      2111      4
: 1987      2112      4
: 1988      2113      4      ! Fill in the DO entry.
: 1989      2114      4      !
: 1990      2115      4      DO_ENTRY [EVENT$L_DO_LIST_COUNT] = 1;
: 1991      2116      4      DO_ENTRY [EVENT$L_DO_LIST_POINT] = .STRING;
: 1992      2117      4      END;
: 1993      2118      4
: 1994      2119      4
: 1995      2120      4      ! Process the /READ, /WRITE, /MODIFY, /EXECUTE, and /RETURN
: 1996      2121      4      ! access adverbs.
: 1997      2122      4      !
: 1998      2123      4      [ADVERB_READ,ADVERB_WRITE,ADVERB_MODIFY,
: 1999      2124      4      ADVERB_EXECUTE,ADVERB_RETURN]:
```



```
.. 2000      2125  4
.. 2001      2126  4
.. 2002      2127  4
.. 2003      2128  4
.. 2004      2129  4
.. 2005      2130  4
.. 2006      2131  4
.. 2007      2132  4
.. 2008      2133  4
.. 2009      2134  4
.. 2010      2135  4
.. 2011      2136  5
.. 2012      2137  5
.. 2013      2138  5
.. 2014      2139  5
.. 2015      2140  5
.. 2016      2141  5
.. 2017      2142  5
.. 2018      2143  5
.. 2019      2144  5
.. 2020      2145  5
.. 2021      2146  5
.. 2022      2147  5
.. 2023      2148  5
.. 2024      2149  5
.. 2025      2150  5
.. 2026      2151  4
.. 2027      2152  4
.. 2028      2153  4
.. 2029      2154  4
.. 2030      2155  4
.. 2031      2156  4
.. 2032      2157  4
.. 2033      2158  4
.. 2034      2159  4
.. 2035      2160  4
.. 2036      2161  4
.. 2037      2162  4
.. 2038      2163  4
.. 2039      2164  4
.. 2040      2165  4
.. 2041      2166  4
.. 2042      2167  4
.. 2043      2168  4
.. 2044      2169  4
.. 2045      2170  4
.. 2046      2171  4
.. 2047      2172  4
.. 2048      2173  4
.. 2049      2174  4
.. 2050      2175  4
.. 2051      2176  4
.. 2052      2177  4
.. 2053      2178  4
.. 2054      2179  4
.. 2055      2180  4
.. 2056      2181  4
```

BEGIN

! Define the command kind to be access.

EVENT_ENTRY [EVENT\$B_CMD_KIND] = EVENT\$K_KIND_ACC;

! Define the sub-kind as appropriate.

```
EVENT_ENTRY [EVENT$B_SUB_KIND] =
  (SELECTONE .ADVERB_NODE [DBG$B_ADVERB_LITERAL] OF
    SET
      [ADVERB_READ]:
        EVENT$K_ACC_READ;
      [ADVERB_WRITE]:
        EVENT$K_ACC_WRIT;
      [ADVERB_MODIFY]:
        EVENT$K_ACC_MDFY;
      [ADVERB_EXECUTE]:
        EVENT$K_ACC_EXEC;
      [ADVERB_RETURN]:
        EVENT$K_ACC_RTRN;
      [OTHERWISE]:
        $DBG_ERROR('DBGEVENT\EVENT_SEMANTICS 20');
    TES
```

);
END;

! Process the /EXCEPTION adverb.

[ADVERB_EXCEPTION]:

BEGIN

EVENT_ENTRY [EVENT\$B_CMD_KIND] = EVENT\$K_KIND_EXC;

EVENT_ENTRY [EVENT\$B_SUB_KIND] = EVENT\$K_EXC_EXC;

END;

! Process the /CALL, /RETURN, /BRANCH, /OPCODE, /INSTRUCTION and
! /LINE instruction-kind adverbs.

```
[ADVERB_CALL,
ADVERB_RETURN,
ADVERB_BRANCH,
ADVERB_OPCODE,
ADVERB_INSTRUCTION,
ADVERB_LINE]:
```

BEGIN

LOCAL

```
  LINENO,      ! Line number returned from PC TO LINE
  MODRST,      ! Module RST returned from PC TO LINE
  STATUS,      ! Status returned from PC TO LINE
  STMTNO;      ! Statement number returned from PC TO LINE
```

! Define the event kind to be instruction.


```
. 2057      2182      4
. 2058      2183      4
. 2059      2184      4
. 2060      2185      4
. 2061      2186      4
. 2062      2187      4
. 2063      2188      5
. 2064      2189      5
. 2065      2190      5
. 2066      2191      5
. 2067      2192      5
. 2068      2193      5
. 2069      2194      5
. 2070      2195      5
. 2071      2196      5
. 2072      2197      5
. 2073      2198      5
. 2074      2199      5
. 2075      2200      5
. 2076      2201      5
. 2077      2202      5
. 2078      2203      4
. 2079      2204      4
. 2080      2205      4
. 2081      2206      4
. 2082      2207      4
. 2083      2208      4
. 2084      2209      4
. 2085      2210      4
. 2086      2211      4
. 2087      2212      4
. 2088      2213      4
. 2089      2214      4
. 2090      2215      4
. 2091      2216      4
. 2092      2217      4
. 2093      2218      4
. 2094      2219      4
. 2095      2220      4
. 2096      2221      4
. 2097      2222      4
. 2098      2223      4
. 2099      2224      4
. 2100      2225      4
. 2101      2226      4
. 2102      2227      4
. 2103      2228      4
. 2104      2229      4
. 2105      2230      4
. 2106      2231      4
. 2107      2232      4
. 2108      2233      4
. 2109      2234      4
. 2110      2235      4
. 2111      2236      4
. 2112      2237      4
. 2113      2238      5
```

```
EVENT_ENTRY [EVENT$B_CMD_KIND] = EVENT$K_KIND_INS;
```

```
! Define the subkind according to the adverb.
```

```
EVENT_ENTRY [EVENT$B_SUB_KIND] =  
  (SELECTONE .ADVERB_NODE [DBG$B_ADVERB_LITERAL] OF  
    SET  
      [ADVERB_CALL]:  
        EVENT$K_INS_CALL;  
      [ADVERB_BRANCH]:  
        EVENT$K_INS_BRAN;  
      [ADVERB_LINE]:  
        EVENT$K_INS_LINE;  
      [ADVERB_OPCODE]:  
        EVENT$K_INS_USER;  
      [ADVERB_INSTRUCTION]:  
        EVENT$K_INS_EVR;  
      [OTHERWISE]:  
        $DBG_ERROR('DBGEVENT\EVENT_SEMANTICS 30');  
    TES  
  );
```

```
! Set the line stepping flag appropriately.
```

```
EVENT_ENTRY [EVENT$V_STEP_LINE] =  
  (.EVENT_ENTRY [EVENT$B_SUB_KIND] EQLU EVENT$K_INS_LINE);
```

```
! We're not in the middle of a step over....
```

```
EVENT_ENTRY [EVENT$V_STEP_BPT] = FALSE;
```

```
! Initialize the LO and HI PC values by calling PC_TO_LINE  
! with the current PC. This gives us a PC range for the  
! line number containing the current PC. This PC range  
! will be used later to determine when a step by line ends.  
! If PC_TO_LINE returns FALSE we are not in a line. In that  
! case, initialize LO and HI PC to 1 and 0, so we will  
! immediately be out of the "range" the next time we check.
```

```
STATUS = 0;  
MODRST = 0;  
IF .EVENT_ENTRY [EVENT$B_SUB_KIND] EQLU EVENT$K_INS_LINE  
THEN  
  STATUS = DBG$PC_TO_LINE_LOOKUP(  
    .DBG$R0NFRAME[DBG$L_USER_PC],  
    LINENO,  
    STMTNO,  
    EVENT_ENTRY[EVENT$L_STEP_LO_PC],  
    EVENT_ENTRY[EVENT$L_STEP_HI_PC],  
    MODRST);  
IF NOT .STATUS  
THEN  
  BEGIN
```



```

: 2114      2239      5
: 2115      2240      5
: 2116      2241      4
: 2117      2242      3
: 2118      2243      3
: 2119      2244      3
: 2120      2245      3
: 2121      2246      3
: 2122      2247      3
: 2123      2248      4
: 2124      2249      4
: 2125      2250      4
: 2126      2251      3
: 2127      2252      3
: 2128      2253      3
: 2129      2254      3
: 2130      2255      3
: 2131      2256      4
: 2132      2257      4
: 2133      2258      4
: 2134      2259      4
: 2135      2260      3
: 2136      2261      3
: 2137      2262      3
: 2138      2263      3
: 2139      2264      3
: 2140      2265      3
: 2141      2266      4
: 2142      2267      4
: 2143      2268      4
: 2144      2269      3
: 2145      2270      3
: 2146      2271      3
: 2147      2272      3
: 2148      2273      3
: 2149      2274      3
: 2150      2275      4
: 2151      2276      4
: 2152      2277      4
: 2153      2278      3
: 2154      2279      3
: 2155      2280      3
: 2156      2281      3
: 2157      2282      3
: 2158      2283      3
: 2159      2284      4
: 2160      2285      4
: 2161      2286      4
: 2162      2287      3
: 2163      2288      3
: 2164      2289      3
: 2165      2290      3
: 2166      2291      3
: 2167      2292      3
: 2168      2293      4
: 2169      2294      4
: 2170      2295      4

      EVENT_ENTRY[EVENT$V_STEP_LO_PC] = 1;
      EVENT_ENTRY[EVENT$V_STEP_HI_PC] = 0;
      END;

      ! Process the /SOURCE adverb.
      !
      [ADVERB_SOURCE]:
      BEGIN
      EVENT_ENTRY [EVENT$V_STEP_SOURCE] = TRUE;
      EVENT_ENTRY [EVENT$V_OVRD_SOURCE] = TRUE;
      END;

      ! Process the /NOSOURCE adverb.
      !
      [ADVERB_NOSOURCE]:
      BEGIN
      EVENT_ENTRY [EVENT$V_STEP_SOURCE] = FALSE;
      EVENT_ENTRY [EVENT$V_OVRD_SOURCE] = TRUE;
      END;

      ! Process the /SYSTEM adverb.
      !
      [ADVERB_SYSTEM]:
      BEGIN
      EVENT_ENTRY [EVENT$V_STEP_NOSYSTEM] = FALSE;
      EVENT_ENTRY [EVENT$V_OVRD_NOSYSTEM] = TRUE;
      END;

      ! Process the /NOSYSTEM adverb.
      !
      [ADVERB_NOSYSTEM]:
      BEGIN
      EVENT_ENTRY [EVENT$V_STEP_NOSYSTEM] = TRUE;
      EVENT_ENTRY [EVENT$V_OVRD_NOSYSTEM] = TRUE;
      END;

      ! Process the /INTO adverb.
      !
      [ADVERB_INT0]:
      BEGIN
      EVENT_ENTRY [EVENT$V_STEP_OVER] = FALSE;
      EVENT_ENTRY [EVENT$V_OVRD_OVER] = TRUE;
      END;

      ! Process the /OVER adverb.
      !
      [ADVERB_OVER]:
      BEGIN
      EVENT_ENTRY [EVENT$V_STEP_OVER] = TRUE;
      EVENT_ENTRY [EVENT$V_OVRD_OVER] = TRUE;
```

```
2171      2296      END;
2172      2297
2173      2298
2174      2299      ! An invalid adverb literal - yell !
2175      2300
2176      2301      [OTHERWISE]:
2177      2302      $DBG_ERROR('DBGEVENT\EVENT_SEMANTICS 40');
2178      2303
2179      2304      TES;
2180      2305
2181      2306
2182      2307      ! Point to the next adverb in the list.
2183      2308
2184      2309      ADVERB_NODE = .ADVERB_NODE [DBG$L_ADVERB_LINK];
2185      2310      END;
2186      2311
2187      2312
2188      2313      ! Now, based on the command kind (access or instruction),
2189      2314      ! process the noun node list.
2190      2315
2191      2316      SELECTONE .EVENT_ENTRY [EVENT$B_CMD_KIND] OF
2192      2317      SET
2193      2318
2194      2319
2195      2320      ! Access events....
2196      2321
2197      2322      [EVENT$K_KIND_ACC]:
2198      2323
2199      2324      IF .EVENT_ENTRY [EVENT$B_CMD_TYPE] NEQU EVENT$K_TYPE_STEPS
2200      2325      THEN
2201      2326      BEGIN
2202      2327
2203      2328      LOCAL
2204      2329      SYMID_LIST; ! Pointer to SYMID
2205      2330
2206      2331
2207      2332      ! Make sure there's a noun node present.
2208      2333
2209      2334      IF .NOUN_NODE EQLA 0
2210      2335      THEN
2211      2336      $DBG_ERROR('DBGEVENT\EVENT_SEMANTICS 50');
2212      2337
2213      2338
2214      2339      ! Build duplicate event entries, one for each
2215      2340      ! address found in the noun node list.
2216      2341
2217      2342      WHILE TRUE DO
2218      2343      BEGIN
2219      2344
2220      2345
2221      2346      ! Obtain the primary descriptor.
2222      2347
2223      2348      PRIMARY = .NOUN_NODE [DBG$L_NOUN_VALUE];
2224      2349
2225      2350
2226      2351      ! Obtain the symid list, and lock it.
2227      2352
```



```
: 2228      2353  4
: 2229      2354  4
: 2230      2355  4
: 2231      2356  4
: 2232      2357  4
: 2233      2358  4
: 2234      2359  4
: 2235      2360  4
: 2236      2361  4
: 2237      2362  4
: 2238      2363  4
: 2239      2364  4
: 2240      2365  4
: 2241      2366  4
: 2242      2367  4
: 2243      2368  4
: 2244      2369  4
: 2245      2370  4
: 2246      2371  4
: 2247      2372  4
: 2248      2373  4
: 2249      2374  4
: 2250      2375  4
: 2251      2376  4
: 2252      2377  4
: 2253      2378  4
: 2254      2379  4
: 2255      2380  4
: 2256      2381  4
: 2257      2382  5
: 2258      2383  5
: 2259      2384  4
: 2260      2385  5
: 2261      2386  4
: 2262      2387  5
: 2263      2388  6
: 2264      2389  5
: 2265      2390  6
: 2266      2391  6
: 2267      2392  5
: 2268      2393  5
: 2269      2394  4
: 2270      2395  4
: 2271      2396  4
: 2272      2397  4
: 2273      2398  4
: 2274      2399  4
: 2275      2400  4
: 2276      2401  4
: 2277      2402  4
: 2278      2403  4
: 2279      2404  4
: 2280      2405  4
: 2281      2406  4
: 2282      2407  4
: 2283      2408  4
: 2284      2409  4
```

```
DBG$NGET_SYMID (.PRIMARY, SYMID_LIST, DUMMY);
DBG$STA_COCK_SYMID (.SYMID_LIST);
```

```
! Save the primary descriptor by making a copy of the primary
! descriptor out of permanent storage.
```

```
DBG$NCOPY_DESC (.PRIMARY, EVENT_ENTRY[EVENT$L_PRIMARY],
                DUMMY, TRUE);
```

```
! Now, get and save the VMS descriptor. (We do this
! after saving the Primary since PRIM_TO_VAL sometimes
! modifies the Primary.)
```

```
DBG$PRIM_TO_VAL (.PRIMARY, DBG$K_V_VALUE_DESC, VALUE);
CH$MOVE T12, VALUE [DBG$A_VALUE_VMSDESC];
EVENT_ENTRY[EVENT$A_VMSDESC];
```

```
! Next, we determine the address where we put the
! breakpoint. We pass our Primary into DBG$NGET_ADDRESS.
! For routines and entry points, NGET_ADDRESS takes a
! flag that says to return the "break address" instead
! of the start address. For routines, the break address
! is after the routine prolog (or, if there is no prolog,
! it is at routine_start+2).
```

```
EVENT_ENTRY [EVENT$V_BREAK_ADDRESS] = FALSE;
IF ((.EVENT_ENTRY [EVENT$B_CMD_TYPE] EQL EVENT$K_TYPE_BREAK) OR
    (.EVENT_ENTRY [EVENT$B_CMD_TYPE] EQL EVENT$K_TYPE_TRACE))
    AND
    (.EVENT_ENTRY [EVENT$B_SUB_KIND] NEQ EVENT$K_ACC_RTRN)
THEN
    BEGIN
        IF (.PRIMARY[DBG$B_DHDR_TYPE] EQL DBG$K_PRIMARY_DESC)
            AND
            ((.PRIMARY[DBG$B_DHDR_KIND] EQL RST$K_ROUTINE) OR
             (.PRIMARY[DBG$B_DHDR_KIND] EQL RST$K_ENTRY))
        THEN
            EVENT_ENTRY [EVENT$V_BREAK_ADDRESS] = TRUE;
    END;
```

```
! Get the address where we place the breakpoint.
```

```
IF NOT DBG$NGET_ADDRESS(.PRIMARY, ADDRESS[0], ADR_TYP,
    .EVENT_ENTRY[EVENT$V_BREAK_ADDRESS], .MESSAGE_VECT)
THEN
    RETURN ST$K_SEVERE;
EVENT_ENTRY [EVENT$L_ADDRESS] = .ADDRESS[0];
```

```
! We may still need to adjust the address of the breakpoint
! by +2 to get past the entry mask of a routine. This
! will be the case if the user set a break on an absolute
! address which happens to be the start of a routine,
```

```
: 2285      2410      4
: 2286      2411      4
: 2287      2412      4
: 2288      2413      4
: 2289      2414      4
: 2290      2415      4
: 2291      2416      4
: 2292      2417      4
: 2293      2418      5
: 2294      2419      5
: 2295      2420      5
: 2296      2421      5
: 2297      2422      5
: 2298      2423      5
: 2299      2424      5
: 2300      2425      5
: 2301      2426      5
: 2302      2427      5
: 2303      2428      5
: 2304      2429      5
: 2305      2430      6
: 2306      2431      6
: 2307      2432      6
: 2308      2433      6
: 2309      2434      6
: 2310      2435      6
: 2311      2436      6
: 2312      2437      6
: 2313      2438      6
: 2314      2439      7
: 2315      2440      6
: 2316      2441      6
: 2317      2442      5
: 2318      2443      4
: 2319      2444      4
: 2320      2445      4
: 2321      2446      4
: 2322      2447      4
: 2323      2448      4
: 2324      2449      5
: 2325      2450      4
: 2326      2451      4
: 2327      2452      4
: 2328      2453      4
: 2329      2454      4
: 2330      2455      4
: 2331      2456      4
: 2332      2457      4
: 2333      2458      4
: 2334      2459      4
: 2335      2460      5
: 2336      2461      5
: 2337      2462      5
: 2338      2463      5
: 2339      2464      5
: 2340      P 2465      5
: 2341      P 2466      5
```

```
: or on data of type ZEM, BPV, or BLV (e.g., if
: the address of a routine is passed as a parameter and
: and a breakpoint is set on that passed-in routine, then
: the Primary describe data of type BPV). These cases
: are not covered by the above "break address" code.
IF NOT .EVENT_ENTRY [EVENT$V_BREAK_ADDRESS]
THEN
  BEGIN

    : If we can determine from the static address table
    : that this is an entry point, then add two to get
    : past the entry mask.
    IF DBG$IS_IT_ENTRY (.ADDRESS [0])
    THEN
      EVENT_ENTRY [EVENT$L_ADDRESS] = .ADDRESS[0] + 2
    ELSE
      BEGIN

        : If the data type in the value descriptor says
        : that we have an entry mask, then add two to
        : get past the entry mask.
        IF (.VALUE[DBG$B_VALUE_DTYPE] EQL DSC$K_DTYPE_ZEM) OR
           (.VALUE[DBG$B_VALUE_DTYPE] EQL DSC$K_DTYPE_BPV) OR
           (.VALUE[DBG$B_VALUE_DTYPE] EQL DSC$K_DTYPE_BLV)
        THEN
          EVENT_ENTRY [EVENT$L_ADDRESS] = .ADDRESS[0] + 2;
        END;
      END;

    : Make sure that we can access the value indicated.
    DBG$READ_ACCESS (.EVENT_ENTRY [EVENT$L_ADDRESS],
      (DBG$DATA_LENGTH(EVENT_ENTRY[EVENT$A_VMSDESC]) + 7)
      ^ -3);

    : Process the event's address additionally based on the
    : access type.
    SELECTONE .EVENT_ENTRY [EVENT$B_SUB_KIND] OF
    SET
      [EVENT$K_ACC_EXEC,
      EVENT$K_ACC_RTRN]:
      BEGIN
        LOCAL
          EVENT_ADDRESS : REF VECTOR [, BYTE];

          EVENT_ADDRESS = .EVENT_ENTRY [EVENT$L_ADDRESS];
          IF NOT WRITE_OPCODE (.EVENT_ADDRESS,
            .EVENT_ADDRESS [0]
```



```
2342 2467 6
2343 2468 5
2344 2469 6
2345 2470 6
2346 2471 6
2347 2472 5
2348 2473 5
2349 2474 5
2350 2475 5
2351 2476 5
2352 2477 5
2353 2478 5
2354 2479 5
2355 2480 5
2356 2481 5
2357 2482 5
2358 2483 4
2359 2484 4
2360 2485 4
2361 2486 4
2362 2487 4
2363 2488 4
2364 2489 4
2365 2490 4
2366 2491 4
2367 2492 5
2368 2493 5
2369 2494 5
2370 2495 5
2371 2496 5
2372 2497 5
2373 2498 5
2374 2499 5
2375 2500 5
2376 2501 5
2377 2502 5
2378 2503 5
2379 2504 5
2380 2505 5
2381 2506 5
2382 2507 5
2383 2508 5
2384 2509 5
2385 2510 5
2386 2511 5
2387 2512 5
2388 2513 5
2389 2514 5
2390 2515 5
2391 2516 5
2392 2517 5
2393 2518 5
2394 2519 5
2395 2520 5
2396 2521 5
2397 2522 5
2398 2523 5
```

```
)
THEN
BEGIN
  SIGNAL (DBG$NOACCESSW, 1, .EVENT_ADDRESS);
  RETURN ST$K_SEVERE;
END;
```

```
! Mark the fact we are at /RET .PC, for we want
! to take the RET SS$ROPRAND fault starting
! from current PC. Normally, we won't take
! the event if we are at current PC.
```

```
IF .EVENT_ADDRESS EQL .DBG$RUNFRAME[DBG$L_USER_PC]
THEN
  EVENT_ENTRY [EVENT$V_RET_AT_PC] = TRUE;
END;
```

```
[EVENT$K_ACC_MDFY]:
```

```
! Process /MODIFY types by building and saving a non-
! volatile value descriptor (i.e. a copy) of the value
! indicated.
```

```
BEGIN
LOCAL
  PRIMARY: REF DBG$PRIMARY;
```

```
! First, get the value descriptor.
! Before we call PRIM_TO_VAL, however, we check
! whether the Primary represents an aggregate,
! such as an entire array. The scheme of saving
! away value descriptors does not work for aggregates,
! because we currently do not represent arrays
! and records with value descriptors. So, for now
! we signal an error saying that we do
! not currently support watchpointing of aggregates.
```

```
DBG$NCOPY_DESC(.EVENT_ENTRY[EVENT$L_PRIMARY],
  PRIMARY, DUMMY, FALSE);
```

```
IF .PRIMARY[DBG$V_DHDR_AGGR]
THEN
  SIGNAL(DBG$NOWATAGGR);
```

```
! In order to keep the code path consistency
! it is better to turn the volatile value
! descriptor into value descriptor.
```

```
DBG$PRIM_TO_VAL (.PRIMARY,
  DBG$K_V_VALUE_DESC,
  VALUE);
DBG$PRIM_TO_VAL (.VALUE,
  DBG$K_VALUE_DESC,
  VALUE)
```

```
2399 2524 5
2400 2525 5
2401 2526 5
2402 2527 5
2403 2528 5
2404 2529 5
2405 2530 5
2406 2531 6
2407 2532 6
2408 2533 6
2409 2534 6
2410 2535 6
2411 2536 6
2412 2537 6
2413 2538 6
2414 2539 6
2415 2540 6
2416 2541 6
2417 2542 6
2418 2543 7
2419 2544 7
2420 2545 7
2421 2546 7
2422 2547 6
2423 2548 6
2424 2549 6
2425 2550 6
2426 2551 6
2427 2552 6
2428 2553 6
2429 2554 6
2430 2555 6
2431 2556 6
2432 2557 7
2433 2558 7
2434 2559 7
2435 2560 6
2436 2561 6
2437 2562 7
2438 2563 6
2439 2564 7
2440 2565 7
2441 2566 6
2442 2567 7
2443 2568 7
2444 2569 7
2445 2570 7
2446 2571 6
2447 2572 6
2448 2573 6
2449 2574 6
2450 2575 6
2451 2576 6
2452 2577 6
2453 2578 6
2454 2579 6
2455 2580 6
```

```
);
IF .VALUE [DBG$B_DHDR_TYPE] NEQU DBG$K_VALUE_DESC
THEN
    SIGNAL(DBG$_WATCHSIZE)
ELSE
    BEGIN
        LOCAL
            PAGE_ENTRY : REF EVENT$PAGE_DESCRIPTOR,
            PAGE_LIST : REF VECTOR [, [LONG],
            PAGE_FOUND;

        ! The user is not permitted to set a watchpoint in
        ! P1 space (or in system space)
        IF .EVENT_ENTRY [EVENT$L_ADDRESS] GEQA P1_SPACE
        THEN
            BEGIN
                SIGNAL(DBG$_BADWATCH, 1,
                    .EVENT_ENTRY[EVENT$L_ADDRESS]);
                RETURN ST$K_SEVERE;
            END;

        ! The user is not permitted to set a watchpoint on
        ! a variable bound to either the context registers
        ! or the user runframe registers.
        IF DBG$MAP TO REG ADDR
            (.EVENT_ENTRY [EVENT$L_ADDRESS], DUMMY)
        THEN
            BEGIN
                SIGNAL(DBG$_NOWATTAR);
                RETURN ST$K_SEVERE;
            END;

        IF (DBG$RUNFRAME [DBG$L_USER_REGS] LEQA
            .EVENT_ENTRY [EVENT$L_ADDRESS]) AND
            (.EVENT_ENTRY [EVENT$L_ADDRESS] LSSA
            DBG$RUNFRAME [DBG$L_USER_REGS] + 68)
        THEN
            BEGIN
                SIGNAL(DBG$_BADWATCH, 1,
                    .EVENT_ENTRY[EVENT$L_ADDRESS]);
                RETURN ST$K_SEVERE;
            END;

        ! Save the value descriptor by making a copy of
        ! the value descriptor out of non-volatile storage.
        ! Obtain the symid list, and lock it.
        DBG$NGET SYMID (.VALUE, SYMID_LIST, DUMMY);
        DBG$STA_LOCK_SYMID (.SYMID_LIST);
```


2456	2581	6
2457	2582	6
2458	2583	6
2459	2584	6
2460	2585	6
2461	2586	6
2462	2587	6
2463	2588	6
2464	2589	6
2465	2590	6
2466	2591	6
2467	2592	6
2468	2593	6
2469	2594	6
2470	2595	6
2471	2596	6
2472	2597	6
2473	2598	6
2474	2599	6
2475	2600	6
2476	2601	6
2477	2602	7
2478	2603	7
2479	2604	7
2480	2605	7
2481	2606	6
2482	2607	6
2483	2608	6
2484	2609	6
2485	2610	6
2486	2611	6
2487	2612	6
2488	2613	7
2489	2614	7
2490	2615	7
2491	2616	7
2492	2617	7
2493	2618	7
2494	2619	7
2495	2620	7
2496	2621	7
2497	2622	8
2498	2623	8
2499	2624	8
2500	2625	8
2501	2626	8
2502	2627	8
2503	2628	8
2504	2629	8
2505	2630	8
2506	2631	8
2507	2632	9
2508	2633	9
2509	2634	9
2510	2635	9
2511	2636	9
2512	2637	9

Page 67
(9)

```

! Copy the value descriptor.
DBG$NCOPY_DESC (.VALUE,
                EVENT_ENTRY [EVENT$L_VALDESC],
                DUMMY);

! Flag that this event has a value descriptor, and
! point the event entry to it's value.
EVENT_ENTRY [EVENT$V_HAS_VAL_DSCR] = TRUE;

! Get the pages associated with this value
IF NOT DBG$NGET PAGES
    (.EVENT_ENTRY [EVENT$L_PRIMARY],
     PAGE_LIST, .MESSAGE_VECT)
THEN
    BEGIN
        SIGNAL (DBG$ BADWATCH, 1,
                .EVENT_ENTRY [EVENT$L_ADDRESS]);
        RETURN ST$SK_SEVERE;
    END;

! While there are entries in the page list, add a
! new page reference in the page queue.
DO
    BEGIN

        ! Search through the existing entries in the
        ! page queue for this page.
        PAGE_ENTRY = .EVENT$PAGE_QUEUE [L_QUEUE_FLINK];
        PAGE_FOUND = FALSE;
        WHILE .PAGE_ENTRY NEQA EVENT$PAGE_QUEUE DO
            BEGIN
                IF .PAGE_ENTRY [EVENT$L_PAGE_ADDRESS] EQ LA
                    .PAGE_LIST [1]
                THEN

                    ! A match is found, so bump the refer-
                    ! ence count, flag a find, and exit
                    ! this loop.
                    BEGIN
                        PAGE_ENTRY [EVENT$W_PAGE_REF_COUNT] =
                            .PAGE_ENTRY [EVENT$W_PAGE_REF_COUNT] + 1;
                        PAGE_FOUND = TRUE;
                        EXIT COOP;
                    END
                END
            END
        END
    END
END

```

```
.. 2513      2638      9
.. 2514      2639      9
.. 2515      2640      9
.. 2516      2641      9
.. 2517      2642      8
.. 2518      2643      8
.. 2519      2644      8
.. 2520      2645      7
.. 2521      2646      7
.. 2522      2647      7
.. 2523      2648      7
.. 2524      2649      7
.. 2525      2650      7
.. 2526      2651      7
.. 2527      2652      7
.. 2528      2653      8
.. 2529      2654      8
.. 2530      2655      8
.. 2531      2656      8
.. 2532      2657      8
.. 2533      2658      8
.. 2534      2659      8
.. 2535      2660      8
.. 2536      2661      8
.. 2537      2662      8
.. 2538      2663      9
.. 2539      2664      8
.. 2540      2665      9
.. 2541      2666      9
.. 2542      2667      9
.. 2543      2668      8
.. 2544      2669      8
.. 2545      2670      8
.. 2546      2671      9
.. 2547      2672      8
.. 2548      2673      9
.. 2549      2674      9
.. 2550      2675      9
.. 2551      2676      8
.. 2552      2677      8
.. 2553      2678      8
.. 2554      2679      8
.. 2555      2680      8
.. 2556      2681      8
.. 2557      2682      7
.. 2558      2683      7
.. 2559      2684      7
.. 2560      2685      6
.. 2561      2686      5
.. 2562      2687      4
.. 2563      2688      4
.. 2564      2689      4
.. 2565      2690      4
.. 2566      2691      4
.. 2567      2692      4
.. 2568      2693      4
.. 2569      2694      4
```

```
! Not found yet, point to the next one.
ELSE
    PAGE_ENTRY = .PAGE_ENTRY [EVENT$PAGE_FLINK];
END;

! If no match was found, add a new entry to the
! page queue, and fill it in.
IF NOT .PAGE_FOUND
THEN
    BEGIN
    LOCAL
        ADDRESS : VECTOR [2],
        PROTECT;

        ADDRESS [0] = .PAGE_LIST [1];
        ADDRESS [1] = .PAGE_LIST [1];
        IF NOT $SETPRT (INADR = ADDRESS,
                        PROT = PRT$C_UR,
                        PRVPRT = PROTECT
                        )
        THEN
            BEGIN
                SIGNAL (DBG$BADWATCH, 1, .PAGE_LIST [1]);
                RETURN ST$K_SEVERE;
            END;
        IF NOT $SETPRT (INADR = ADDRESS,
                        PROT = .PROTECT
                        )
        THEN
            BEGIN
                SIGNAL (DBG$NOWPROT);
                RETURN ST$K_SEVERE;
            END;
        PAGE_ENTRY = DBG$GET_MEMORY (EVENT$K_PAGE_DESCRIPTOR_SIZE);
        INSQUE (.PAGE_ENTRY, EVENT$PAGE_QUEUE);
        PAGE_ENTRY [EVENT$PAGE_ADDRESS] =
            .PAGE_LIST [1];
        PAGE_ENTRY [EVENT$PAGE_REF_COUNT] = 1;
    END;
    PAGE_LIST = .PAGE_LIST [0];
END
WHILE .PAGE_LIST;
END;
END;
TES;

! Point to the next noun node.
NOUN_NODE = .NOUN_NODE [DBG$L_NOUN_LINK];
```



```
.. 2570      2695  4
.. 2571      2696  4
.. 2572      2697  4
.. 2573      2698  4
.. 2574      2699  4
.. 2575      2700  4
.. 2576      2701  4
.. 2577      2702  5
.. 2578      2703  5
.. 2579      2704  5
.. 2580      2705  5
.. 2581      2706  5
.. 2582      2707  5
.. 2583      2708  5
.. 2584      2709  5
.. 2585      2710  5
.. 2586      2711  5
.. 2587      2712  5
.. 2588      2713  5
.. 2589      2714  5
.. 2590      2715  5
.. 2591      2716  5
.. 2592      2717  5
.. 2593      2718  5
.. 2594      2719  5
.. 2595      2720  5
.. 2596      2721  5
.. 2597      2722  5
.. 2598      2723  5
.. 2599      2724  5
.. 2600      2725  5
.. 2601      2726  5
.. 2602      2727  5
.. 2603      2728  5
.. 2604      2729  5
.. 2605      2730  5
.. 2606      2731  5
.. 2607      2732  5
.. 2608      2733  5
.. 2609      2734  5
.. 2610      2735  5
.. 2611      2736  5
.. 2612      2737  5
.. 2613      2738  5
.. 2614      2739  5
.. 2615      2740  5
.. 2616      2741  4
.. 2617      2742  4
.. 2618      2743  4
.. 2619      2744  4
.. 2620      2745  4
.. 2621      2746  2
.. 2622      2747  2
.. 2623      2748  2
.. 2624      2749  2
.. 2625      2750  2
.. 2626      2751  2
```

```
! If there's more addresses, get an new event descriptor
! from permanent memory, copy the current one to it, and
! link it to the working queue; otherwise, exit this loop.
```

```
IF .NOUN_NODE NEQA 0
THEN
```

```
  BEGIN
```

```
    ! Get another event descriptor, and copy the old
    ! one into it.
```

```
    EVENT_ENTRY = DBG$GET MEMORY
                  (EVENT$K_EVENT_DESCRIPTOR_SIZE);
```

```
    CH$MOVE
      (EVENT$K_EVENT_DESCRIPTOR_SIZE * 4,
       .EVENT_QUEUE [L_QUEUE_FLINK],
       .EVENT_ENTRY
      );
```

```
    ! Link the new one into the working queue.
```

```
    INSQUE (.EVENT_ENTRY, EVENT_QUEUE);
```

```
    ! If the old entry has a WHEN, bump the WHEN's
    ! reference count.
```

```
    IF .EVENT_ENTRY [EVENT$L_WHEN] NEQA 0
    THEN
      WHEN_ENTRY [EVENT$L_WHEN_COUNT] =
        .WHEN_ENTRY [EVENT$L_WHEN_COUNT] + 1;
```

```
    ! If the old entry has a DO LIST, bump the DO LIST's
    ! reference count.
```

```
    IF .EVENT_ENTRY [EVENT$L_DO] NEQA 0
    THEN
      DO_ENTRY [EVENT$L_DO_LIST_COUNT] =
        .DO_ENTRY [EVENT$L_DO_LIST_COUNT] + 1;
```

```
    END
```

```
  ELSE
    EXITLOOP;
```

```
  END
```

```
END;
```

```
! Instruction events...
[EVENT$K_KIND_INS]:
```

```
: 2627      2752      2
: 2628      2753      2
: 2629      2754      2
: 2630      2755      2
: 2631      2756      2
: 2632      2757      2
: 2633      2758      2
: 2634      2759      2
: 2635      2760      2
: 2636      2761      2
: 2637      2762      2
: 2638      2763      2
: 2639      2764      2
: 2640      2765      2
: 2641      2766      2
: 2642      2767      2
: 2643      2768      2
: 2644      2769      2
: 2645      2770      2
: 2646      2771      2
: 2647      2772      2
: 2648      2773      2
: 2649      2774      2
: 2650      2775      2
: 2651      2776      2
: 2652      2777      2
: 2653      2778      2
: 2654      2779      2
: 2655      2780      2
: 2656      2781      2
: 2657      2782      2
: 2658      2783      2
: 2659      2784      2
: 2660      2785      2
: 2661      2786      2
: 2662      2787      2
: 2663      2788      2
: 2664      2789      2
: 2665      2790      2
: 2666      2791      2
: 2667      2792      2
: 2668      2793      2
: 2669      2794      2
: 2670      2795      2
: 2671      2796      2
: 2672      2797      2
: 2673      2798      2
: 2674      2799      2
: 2675      2800      2
: 2676      2801      2
: 2677      2802      2
: 2678      2803      2
: 2679      2804      2
: 2680      2805      2
: 2681      2806      2
: 2682      2807      2
: 2683      2808      2
```

```
SELECTONE .EVENT_ENTRY [EVENT$B_SUB_KIND] OF
SET
[EVENT$K_INS_CALL]:
    EVENT_ENTRY [EVENT$L_OPCODE_LIST] = DBG$OPCODES_CALL;
[EVENT$K_INS_BRAN]:
    EVENT_ENTRY [EVENT$L_OPCODE_LIST] = DBG$OPCODES_BRANCH;
[EVENT$K_INS_EVR]:
    EVENT_ENTRY [EVENT$L_OPCODE_LIST] = 0;
[EVENT$K_INS_USER]:
    BEGIN
        LOCAL
            OPCODE_LIST: REF BITVECTOR[512];

        ! Allocate space for the opcode list.
        ! 512 bits = 16 longwords
        OPCODE_LIST = DBG$GET_MEMORY(16);

        ! If this is a STEP or SET STEP command without any /INST=(opcode...)
        ! and we get here then the user must have previously
        ! done SET STEP INST=(opcode-list). In this case we
        ! initialize the OPCODE_LIST from the copy of
        ! the opcode list that we saved away on the SET STEP command.
        ! Otherwise, we just clear the opcode list.
        IF (.NOUN_NODE EQLA 0) AND
            ((.VERB_NODE[DBG$B_VERB_COMPOSITE] EQL EVENT$K_STEP) OR
             (.VERB_NODE[DBG$B_VERB_COMPOSITE] EQL EVENT$K_SET_STEP))
        THEN
            BEGIN
                CH$MOVE (64, DBG$OPCODES_STEP_USER, .OPCODE_LIST);
                CH$MOVE (64, DBG$OPCODES_STEP_USER, DBG$OPCODES_USER);
            END
        ELSE
            BEGIN
                CH$FILL (0, 64, .OPCODE_LIST);
                CH$FILL (0, 64, DBG$OPCODES_USER);
            END;

        ! For each opcode (found in the noun node(s) value,
        ! set the corresponding opcode's bit in the User
        ! opcode bitmap.
        WHILE .NOUN_NODE NEQA 0 DO
            BEGIN
                OPCODE_LIST [.NOUN_NODE [DBG$L_NOUN_VALUE]] = 1;
                DBG$OPCODES_USER [.NOUN_NODE [DBG$L_NOUN_VALUE]] = 1;
                NOUN_NODE = .NOUN_NODE [DBG$L_NOUN_INK];
            END;

        ! Now point the event entry to the user opcode bitmap.
        EVENT_ENTRY [EVENT$L_OPCODE_LIST] = .OPCODE_LIST;
```



```
: 2684      2809      2      END;
: 2685      2810      2
: 2686      2811      2
: 2687      2812      2      [EVENT$K_INS_LINE]:
: 2688      2813      2      EVENT_ENTRY [EVENT$L_OPCODE_LIST] = 0;
: 2689      2814      2
: 2690      2815      2
: 2691      2816      2      ! An invalid sub-kind, yell !
: 2692      2817      2
: 2693      2818      2      [OTHERWISE]:
: 2694      2819      2      $DBG_ERROR('DBGEVENT\EVENT_SEMANTICS 60');
: 2695      2820      2
: 2696      2821      2      TES;
: 2697      2822      2
: 2698      2823      2
: 2699      2824      2      ! /EXCEPTION kind - do nothing.
: 2700      2825      2
: 2701      2826      2      [EVENT$K_KIND_EXC]:
: 2702      2827      2      0;
: 2703      2828      2
: 2704      2829      2      ! An invalid kind, yell !
: 2705      2830      2
: 2706      2831      2      [OTHERWISE]:
: 2707      2832      2      $DBG_ERROR('DBGEVENT\EVENT_SEMANTICS 70');
: 2708      2833      2
: 2709      2834      2      TES;
: 2710      2835      2
: 2711      2836      2
: 2712      2837      2      ! If this is a STEP or SET STEP command, process the entry now.
: 2713      2838      2
: 2714      2839      2      IF .EVENT_ENTRY [EVENT$B_CMD_TYPE] EQLU EVENT$K_TYPE_STEPS
: 2715      2840      2      THEN
: 2716      2841      2      BEGIN
: 2717      2842      2
: 2718      2843      2
: 2719      2844      2      ! If this is a SET STEP command, just point to the user-defined STEP
: 2720      2845      2      definitions.
: 2721      2846      2
: 2722      2847      2      IF .VERB_NODE [DBG$B_VERB_COMPOSITE] EQLU EVENT$K_SET_STEP
: 2723      2848      2      THEN
: 2724      2849      2      DBG$SET_STP_LVL (USER_DEF_STEP)
: 2725      2850      2
: 2726      2851      2
: 2727      2852      2      ! This is a STEP command.
: 2728      2853      2
: 2729      2854      2      ELSE
: 2730      2855      2      BEGIN
: 2731      2856      2
: 2732      2857      2
: 2733      2858      2      ! This is a STEP command. Make sure that we're not trying to
: 2734      2859      2      step from an exception.
: 2735      2860      2
: 2736      2861      2      IF .DBG$GB_EXC_BRE_FLAG
: 2737      2862      2      THEN
: 2738      2863      2      SIGNAL(DBG$_STEFROEXC)
: 2739      2864      2
: 2740      2865      2
```



```
: 2741      2866  4      ! We are not stepping from an exception, so we can indeed step.'
: 2742      2867  4      !
: 2743      2868  4      ELSE
: 2744      2869  5      BEGIN
: 2745      2870  5
: 2746      2871  5
: 2747      2872  5      ! Make sure that we can start up where we are at.
: 2748      2873  5      !
: 2749      2874  5      IF NOT PROBER(%REF(0), %REF(1), .DBG$RUNFRAME[DBG$L_USER_PC])
: 2750      2875  5      THEN
: 2751      2876  5          SIGNAL(DBG$_BADSTARTPC, 1, .DBG$RUNFRAME[DBG$L_USER_PC])
: 2752      2877  5
: 2753      2878  5
: 2754      2879  5      ! We can indeed start here, so this STEP is okay.
: 2755      2880  5      !
: 2756      2881  5      ELSE
: 2757      2882  6      BEGIN
: 2758      2883  6
: 2759      2884  6
: 2760      2885  6      ! Point to STEP override definitions.
: 2761      2886  6      !
: 2762      2887  6      DBG$SET_STP_LVL (OVERRIDE_STEP);
: 2763      2888  6
: 2764      2889  6
: 2765      2890  6      ! Update the step count (probably not needed any more).
: 2766      2891  6      !
: 2767      2892  6      DBG$GL_STEP_NUM = .EVENT_ENTRY [EVENT$L_AFTER_COUNT];
: 2768      2893  6
: 2769      2894  6
: 2770      2895  6      ! Is the user didn't type STEP 0 (a NOP), add this entry
: 2771      2896  6      ! to the event queue.
: 2772      2897  6      !
: 2773      2898  6      IF .EVENT_ENTRY [EVENT$L_AFTER_COUNT] GTRU 0
: 2774      2899  6      THEN
: 2775      2900  7          BEGIN
: 2776      2901  7
: 2777      2902  7
: 2778      2903  7      ! Remove this new entry from the working queue, and
: 2779      2904  7      ! insert it into HEAD end of the Command Queue (we
: 2780      2905  7      ! want steps to show up first).
: 2781      2906  7      !
: 2782      2907  7      REMQUE (.EVENT_ENTRY, EVENT_ENTRY);
: 2783      2908  7      INSQUE (.EVENT_ENTRY, EVENT$CMD_QUEUE);
: 2784      2909  7
: 2785      2910  7
: 2786      2911  7      ! Initialize the queue header within the entry (FLINK1
: 2787      2912  7      ! and BLINK1) to be empty....
: 2788      2913  7      !
: 2789      2914  7      EVENT_ENTRY [EVENT$L_EXC_FLINK] = .EVENT_ENTRY;
: 2790      2915  7      EVENT_ENTRY [EVENT$L_EXC_BLINK] = .EVENT_ENTRY;
: 2791      2916  7
: 2792      2917  7
: 2793      2918  7      ! If this is a STEP/RETURN, save the current PC and
: 2794      2919  7      ! set the saved FP to the current FP (unless we are
: 2795      2920  7      ! looking at a RET instruction, in which case we're
: 2796      2921  7      ! going to STEP out one routine level).
: 2797      2922  7      !
```



```
: 2798      2923  7      IF .EVENT_ENTRY [EVENT$B_SUB_KIND] EQLU
: 2799      2924  7      EVENT$R_ACC_RTRN
: 2800      2925  7      THEN
: 2801      2926  8      BEGIN
: 2802      2927  8      BUILTIN PROBER;
: 2803      2928  8      LOCAL
: 2804      2929  8      CALL FRAME      : REF VECTOR[,LONG],
: 2805      2930  8      STACK_FRAME   : REF BLOCK [,BYTE],
: 2806      2931  8      INSTRUCTION   : BYTE;
: 2807      2932  8
: 2808      2933  8      EVENT_ENTRY [EVENT$L_USERS_PC] = .DBG$RUNFRAME [DBG$L_USER_PC];
: 2809      2934  8      EVENT_ENTRY [EVENT$L_USERS_FP] = .DBG$RUNFRAME [DBG$L_USER_FP];
: 2810      2935  8      IF PROBER(%REF(0),%REF(1),.DBG$RUNFRAME[DBG$L_USER_PC])
: 2811      2936  8      THEN
: 2812      2937  9      BEGIN
: 2813      2938  9      INSTRUCTION = .(.DBG$RUNFRAME[DBG$L_USER_PC])<0,8,0>;
: 2814      2939 10      IF (.INSTRUCTION EQLU %X'04')
: 2815      2940  9      THEN
: 2816      2941 10      BEGIN
: 2817      2942 10      STACK_FRAME = .DBG$RUNFRAME[DBG$L_USER_FP];
: 2818      2943 10      IF PROBER(%REF(0), %REF(20), .STACK_FRAME)
: 2819      2944 10      THEN
: 2820      2945 10      EVENT_ENTRY[EVENT$L_USERS_FP] =
: 2821      2946 10      .STACK_FRAME[SF$L_SAVE_FP];
: 2822      2947  9      END;
: 2823      2948  8      END;
: 2824      2949  8
: 2825      2950  8      ! Save more info from FP. in the Event.
: 2826      2951  8      !
: 2827      2952  8      CALL_FRAME = DBG$GET_MEMORY(5);
: 2828      2953  8      EVENT_ENTRY[EVENT$L_CALL_FRAME] = .CALL_FRAME;
: 2829      2954  8      CH$MOVE(20,.EVENT_ENTRY[EVENT$L_USERS_FP],.CALL_FRAME);
: 2830      2955  7      END;
: 2831      2956  7
: 2832      2957  7
: 2833      2958  7      ! This was a STEP command - stop taking commands.
: 2834      2959  7      !
: 2835      2960  7      DBG$GB_TAKE_CMD = FALSE;
: 2836      2961  6      END;
: 2837      2962  6
: 2838      2963  5      END;
: 2839      2964  5
: 2840      2965  4      END;
: 2841      2966  4
: 2842      2967  3      END;
: 2843      2968  3
: 2844      2969  3
: 2845      2970  3      ! Setup the stepping flags.
: 2846      2971  3      !
: 2847      2972  3      DBG$GB_STP_PTR [EVENT$B STEPPING KIND] =
: 2848      2973  3      .EVENT_ENTRY [EVENT$B SUB_KIND];
: 2849      2974  3      DBG$GB_STP_PTR [EVENT$V STEPPING LINE] =
: 2850      2975  3      .EVENT_ENTRY [EVENT$V STEP LINE];
: 2851      2976  3      DBG$GB_STP_PTR [EVENT$V STEPPING OVER] =
: 2852      2977  3      .EVENT_ENTRY [EVENT$V STEP OVER];
: 2853      2978  3      DBG$GB_STP_PTR [EVENT$V STEPPING SOURCE] =
: 2854      2979  3      .EVENT_ENTRY [EVENT$V_STEP_SOURCE];
```



```
2855 2980 DBG$GB_STP_PTR [EVENT$V STEPPING NOSYSTEM] =
2856 2981 .EVENT_ENTRY [EVENT$V STEP NOSYSTEM];
2857 2982 DBG$GB_STP_PTR [EVENT$V STEPPING NOSILENT] =
2858 2983 .EVENT_ENTRY [EVENT$V STEP NOSILENT];
2859 2984
2860 2985
2861 2986 ! Save away the opcode list we have built. We get here on
2862 2987 SET STEP INST= (opcode-list)
2863 2988
2864 2989 IF .VERB_NODE[DBG$B_VERB_COMPOSITE] EQL EVENT$._SET_STEP
2865 2990 THEN
2866 2991 BEGIN
2867 2992 IF .EVENT_ENTRY[EVENT$B_SUB_KIND] EQL EVENT$K_INS_USER
2868 2993 THEN
2869 2994 BEGIN
2870 2995 CH$MOVE(64, DBG$OPCODES_USER, DBG$OPCODE_STEP_USER);
2871 2996 DBG$GB_STP_PTR [EVENT$L_STEPPING_OP_LIST] = DBG$OPCODES_STEP_USER;
2872 2997 END
2873 2998
2874 2999 ELSE
2875 3000 DBG$GB_STP_PTR [EVENT$L_STEPPING_OP_LIST] = 0;
2876 3001 END;
2877 3002 END
2878 3003
2879 3004 ! This is not a STEP or SET STEP command, so it must be SET BREAK, SET
2880 3005 TRACE, or SET WATCH.
2881 3006
2882 3007 ELSE
2883 3008 BEGIN
2884 3009
2885 3010 ! Now that we've built a working list of event entries, add them to
2886 3011 the Command Queue, deleting existing entries that match new ones.
2887 3012
2888 3013 ! For each new event entry, check the existing entries in the
2889 3014 Command queue for any entries that match (command types, and
2890 3015 addresses if the command type is an access type).
2891 3016
2892 3017 ! So, while there are (still) entries on the working queue....
2893 3018
2894 3019 WHILE .EVENT_QUEUE [L_QUEUE_FLINK] NEQA EVENT_QUEUE DO
2895 3020 BEGIN
2896 3021
2897 3022 ! Point EVENT_ENTRY to the first new event entry on the
2898 3023 working queue.
2899 3024
2900 3025 EVENT_ENTRY = .EVENT_QUEUE [L_QUEUE_FLINK];
2901 3026
2902 3027
2903 3028 ! Point to the first and next existing entries in the Command Queue.
2904 3029
2905 3030 QUEUE_ENTRY = .EVENT$CMD_QUEUE [L_QUEUE_FLINK];
2906 3031 NEXTQ_ENTRY = .QUEUE_ENTRY [EVENT$L_CMD_FLINK];
2907 3032
2908 3033
2909 3034
2910 3035
2911 3036
```



```
.. 2912 3037 4
... 2913 3038 4
... 2914 3039 4
... 2915 3040 5
... 2916 3041 5
... 2917 3042 5
... 2918 3043 5
... 2919 3044 5
... 2920 3045 5
... 2921 3046 5
... 2922 3047 5
... 2923 3048 6
... 2924 3049 6
... 2925 3050 6
... 2926 3051 6
... 2927 3052 6
... 2928 3053 6
... 2929 3054 6
... 2930 3055 6
... 2931 3056 6
... 2932 3057 7
... 2933 3058 7
... 2934 3059 7
... 2935 3060 7
... 2936 3061 7
... 2937 3062 7
... 2938 3063 7
... 2939 3064 7
... 2940 3065 7
... 2941 3066 7
... 2942 3067 7
... 2943 3068 7
... 2944 3069 7
... 2945 3070 7
... 2946 3071 7
... 2947 3072 8
... 2948 3073 8
... 2949 3074 8
... 2950 3075 8
... 2951 3076 8
... 2952 3077 8
... 2953 3078 8
... 2954 3079 8
... 2955 3080 8
... 2956 3081 8
... 2957 3082 8
... 2958 3083 10
... 2959 3084 10
... 2960 3085 10
... 2961 3086 10
... 2962 3087 9
... 2963 3088 9
... 2964 3089 9
... 2965 3090 9
... 2966 3091 9
... 2967 3092 9
.. 2968 3093 9
```

```
! For each existing entry...
WHILE .QUEUE_ENTRY NEQA EVENT$CMD_QUEUE DO
  BEGIN

  ! Ignore any 'deleted' entries that haven't actually been
  ! removed.
  IF NOT .QUEUE_ENTRY [EVENT$V_DELETED]
  THEN
    BEGIN

    ! Compare the command kind of this existing entry and the
    ! new one.
    IF .EVENT_ENTRY [EVENT$B_CMD_KIND] EQLU
      .QUEUE_ENTRY [EVENT$B_CMD_KIND]
    THEN
      BEGIN

      ! The command kinds match. Select on the kind.
      !
      SELECTONE .EVENT_ENTRY [EVENT$B_CMD_KIND] OF
        SET

      ! The command kind is access (/READ, /WRITE,
      ! /MODIFY, /EXECUTE, or /RETURN). We now have to
      ! compare the entries to determine if the existing
      ! one must be deleted.
      [EVENT$K_KIND_ACC]:
      BEGIN
        LOCAL CODE_1, CODE_2;

        ! If either of the entries is /EXECUTE or
        ! /RETURN, we'll compare the byte addresses,
        ! otherwise compare the two VMS descriptors,
        ! and delete the existing entry if they're
        ! the same.
        CODE_1 = .EVENT_ENTRY [EVENT$B_SUB_KIND];
        CODE_2 = .QUEUE_ENTRY [EVENT$B_SUB_KIND];
        IF (IF (.CODE_1 EQLU EVENT$K_ACC_EXEC)
          OR (.CODE_1 EQLU EVENT$K_ACC_RTRN)
          OR (.CODE_2 EQLU EVENT$K_ACC_EXEC)
          OR (.CODE_2 EQLU EVENT$K_ACC_RTRN)
        THEN
          .EVENT_ENTRY [EVENT$L_ADDRESS] EQLA
            .QUEUE_ENTRY [EVENT$L_ADDRESS]
        ELSE
          COMPARE_VMSDESC(.EVENT_ENTRY [EVENT$A_VMSDESC],
            QUEUE_ENTRY [EVENT$A_VMSDESC])
        )
      )
    )
  )
END
```

:	2969	3094	8
:	2970	3095	8
:	2971	3096	7
:	2972	3097	7
:	2973	3098	7
:	2974	3099	7
:	2975	3100	7
:	2976	3101	7
:	2977	3102	7
:	2978	3103	7
:	2979	3104	7
:	2980	3105	7
:	2981	3106	7
:	2982	3107	7
:	2983	3108	7
:	2984	3109	7
:	2985	3110	7
:	2986	3111	7
:	2987	3112	7
:	2988	3113	7
:	2989	3114	8
:	2990	3115	7
:	2991	3116	8
:	2992	3117	8
:	2993	3118	7
:	2994	3119	7
:	2995	3120	7
:	2996	3121	7
:	2997	3122	7
:	2998	3123	7
:	2999	3124	7
:	3000	3125	7
:	3001	3126	7
:	3002	3127	7
:	3003	3128	7
:	3004	3129	7
:	3005	3130	7
:	3006	3131	7
:	3007	3132	7
:	3008	3133	7
:	3009	3134	7
:	3010	3135	7
:	3011	3136	7
:	3012	3137	7
:	3013	3138	6
:	3014	3139	6
:	3015	3140	5
:	3016	3141	5
:	3017	3142	5
:	3018	3143	5
:	3019	3144	5
:	3020	3145	5
:	3021	3146	5
:	3022	3147	4
:	3023	3148	4
:	3024	3149	4
:	3025	3150	4

```
THEN
  QUEUE_ENTRY [EVENT$V_DELETED] = TRUE;
END;
```

```
! The command kind is instruction. Compare the two
! entries' subkind, and delete the existing entry
! if they're the same.
```

```
[EVENT$K_KIND_INS]:
CASE .EVENT_ENTRY [EVENT$B_SUB_KIND]
FROM EVENT$K_INS_CALL TO EVENT$K_INS_USER OF
SET
[EVENT$K_INS_CALL TO EVENT$K_INS_LINE]:
  IF .EVENT_ENTRY [EVENT$B_SUB_KIND] EQLU
    .QUEUE_ENTRY [EVENT$B_SUB_KIND]
  THEN
    QUEUE_ENTRY [EVENT$V_DELETED] =
      TRUE;
[EVENT$K_INS_EVR TO EVENT$K_INS_USER]:
  IF (.QUEUE_ENTRY [EVENT$B_SUB_KIND] EQLU
    EVENT$K_INS_EVR) OR
    (.QUEUE_ENTRY [EVENT$B_SUB_KIND] EQLU
    EVENT$K_INS_USER)
  THEN
    QUEUE_ENTRY [EVENT$V_DELETED] =
      TRUE;
TES;
```

```
! The access type is exception, so delete the
! existing entry.
```

```
[EVENT$K_KIND_EXC]:
  QUEUE_ENTRY [EVENT$V_DELETED] = TRUE;
```

```
! The command kind is invalid - yell !
```

```
[OTHERWISE]:
  $DBG_ERROR('DBGEVENT\EVENT_SEMANTICS 80');
```

```
TES;
```

```
END;
```

```
END;
```

```
! Point to the next Command Queue entries.
```

```
QUEUE_ENTRY = .NEXTQ_ENTRY;
NEXTQ_ENTRY = .QUEUE_ENTRY [EVENT$L_CMD_FLINK];
END;
```

```
! Remove this new entry from the working queue, and insert
```



```
3026      3151  4      ! it into tail end of the Command Queue.
3027      3152  4      !
3028      3153  4      REMQUE (.EVENT_ENTRY, EVENT_ENTRY);
3029      3154  4      INSQUE (.EVENT_ENTRY, .EVENT$CMD_QUEUE [L_QUEUE_BLINK]);
3030      3155  4      !
3031      3156  4      !
3032      3157  4      ! Initialize the queue header within the entry (FLINK1 and BLINK1)
3033      3158  4      ! to be empty.
3034      3159  4      !
3035      3160  4      EVENT_ENTRY [EVENT$L_EXC_FLINK] = .EVENT_ENTRY;
3036      3161  4      EVENT_ENTRY [EVENT$L_EXC_BLINK] = .EVENT_ENTRY;
3037      3162  3      END;
3038      3163  3      !
3039      3164  2      END;
3040      3165  2      !
3041      3166  2      ! Delete any entries that remain in the working queue.
3042      3167  2      ! This happens on SET STEP or on STEP 0.
3043      3168  2      !
3044      3169  2      !
3045      3170  2      EVENT_ENTRY = .EVENT_QUEUE[L_QUEUE_FLINK];
3046      3171  2      WHILE .EVENT_ENTRY NEQU EVENT_QUEUE DO
3047      3172  2      BEGIN
3048      3173  3      LOCAL
3049      3174  3      NEW_EVENT_ENTRY;
3050      3175  3      NEW_EVENT_ENTRY = .EVENT_ENTRY[EVENT$L_CMD_FLINK];
3051      3176  3      DELETE_EVENT_ENTRY(.EVENT_ENTRY);
3052      3177  3      EVENT_ENTRY = .NEW_EVENT_ENTRY;
3053      3178  2      END;
3054      3179  2      !
3055      3180  2      ! All's well that ends well.
3056      3181  2      !
3057      3182  2      RETURN ST$K_SUCCESS;
3058      3183  2      !
3059      3184  2      !
3060      3185  1      END;
```

```
54 4E 45 56 45 5C 54 4E 45 56 45 47 42 44 1B 000C1 P.ABL: .ASCII <27>\DBGEVENT\<92>\EVENT_SEMANTICS 10\
54 4E 45 56 45 5C 54 4E 45 56 45 47 42 44 1B 000D0 P.ABM: .ASCII <27>\DBGEVENT\<92>\EVENT_SEMANTICS 20\
54 4E 45 56 45 5C 54 4E 45 56 45 47 42 44 1B 000EC P.ABN: .ASCII <27>\DBGEVENT\<92>\EVENT_SEMANTICS 30\
54 4E 45 56 45 5C 54 4E 45 56 45 47 42 44 1B 00108 P.ABO: .ASCII <27>\DBGEVENT\<92>\EVENT_SEMANTICS 40\
54 4E 45 56 45 5C 54 4E 45 56 45 47 42 44 1B 00115 P.ABP: .ASCII <27>\DBGEVENT\<92>\EVENT_SEMANTICS 50\
54 4E 45 56 45 5C 54 4E 45 56 45 47 42 44 1B 00124 P.ABQ: .ASCII <27>\DBGEVENT\<92>\EVENT_SEMANTICS 60\
54 4E 45 56 45 5C 54 4E 45 56 45 47 42 44 1B 00131 P.ABR: .ASCII <27>\DBGEVENT\<92>\EVENT_SEMANTICS 70\
54 4E 45 56 45 5C 54 4E 45 56 45 47 42 44 1B 00140 P.ABS: .ASCII <27>\DBGEVENT\<92>\EVENT_SEMANTICS 80\
54 4E 45 56 45 5C 54 4E 45 56 45 47 42 44 1B 0015C
54 4E 45 56 45 5C 54 4E 45 56 45 47 42 44 1B 00169
54 4E 45 56 45 5C 54 4E 45 56 45 47 42 44 1B 00178
54 4E 45 56 45 5C 54 4E 45 56 45 47 42 44 1B 00185
54 4E 45 56 45 5C 54 4E 45 56 45 47 42 44 1B 00194
```

				OFFC 00000			
		5E	B4	AE	9E	00002	.EXTRN SYSS\$SETPRT
	3C	AE	3C	AE	9E	00006	.PSECT DBG\$CODE,NOWRT, SHR, PIC,0
	40	AE	3C	AE	9E	0000B	.ENTRY DBG\$EVENT SEMANTICS, Save R2,R3,R4,R5,R6,-
				10	DD	00010	R7,R8,R9,R10,R11
00000000G		00		01	FB	00012	-76(SP), SP
		56		50	DD	00019	MOVAB EVENT_QUEUE, EVENT_QUEUE
	3C	AE		66	0E	0001C	MOVAB EVENT_QUEUE, EVENT_QUEUE+4
	08	A6		56	DD	00020	PUSHL #16
	0C	A6		56	DD	00024	CALLS #1, DBG\$GET MEMORY
		52	04	AC	DD	00028	MOVL R0, EVENT_ENTRY
		50	01	A2	9A	0002C	INSQUE (EVENT_ENTRY), EVENT_QUEUE
		01		50	91	00030	MOVL EVENT_ENTRY, 8(EVENT_ENTRY)
				05	12	00033	MOVL EVENT_ENTRY, 12(EVENT_ENTRY)
			14	A6	94	00035	MOVL VERB_NODE, R2
				4C	11	00038	MOVZBL 1(R2), R0
		03		50	91	0003A	CMPB R0, #1
				08	12	0003D	BNEQ 1\$
	14	A6	0200	8F	B0	0003F	CLRB 20(EVENT_ENTRY)
				3F	11	00045	BRB 6\$
		0E		50	91	00047	CMPB R0, #3
				06	12	0004A	BNEQ 2\$
	14	A6		01	90	0004C	MOVW #512, 20(EVENT_ENTRY)
				34	11	00050	BRB 6\$
		11		50	91	00052	CMPB R0, #14
				0A	12	00055	BNEQ 3\$
	14	A6		02	B0	00057	MOVW #1, 20(EVENT_ENTRY)
	16	A6		02	90	0005B	BRB 6\$
				25	11	0005F	MOVW #2, 20(EVENT_ENTRY)
		0B		50	91	00061	MOVW #2, 22(EVENT_ENTRY)
				0B	1F	00064	BRB 6\$
		0C		50	91	00066	CMPB R0, #11
				06	1A	00069	BLSSU 5\$
	14	A6		03	90	0006B	CMPB R0, #12
				15	11	0006F	BGTRU 5\$
			00000000'	EF	9F	00071	MOVW #3, 20(EVENT_ENTRY)
				01	DD	00077	BRB 6\$
			00028362	8F	DD	00079	PUSHAB P.ABL
00000000G		00		03	FB	0007F	PUSHL #1
				A6	94	00086	PUSHL #164706
		50		A6	9E	00089	CALLS #3, LIB\$SIGNAL
				60	D4	0008D	CLRB 23(EVENT_ENTRY)
		03		A6	91	0008F	MOVAB 24(EVENT_ENTRY), R0
				3A	12	00093	CLRL (R0)
		51	00000000G	00	DD	00095	CMPB 20(EVENT_ENTRY), #3
		01		A1	FD	0009C	BNEQ 7\$
60	01	01		09	EF	000A2	MOVL DBG\$GB_STP_PTR, R1
53	61	03		53	FD	000A7	INSV 1(R1), #1, #1, (R0)
60	01	01		0A	EF	000AC	EXTZV #9, #1, (R1), R3
53	61	05		53	FD	000B1	INSV R3, #3, #1, (R0)
60	01	60		10	8A	000B6	EXTZV #10, #1, (R1), R3
				0B	EF	000B9	INSV R3, #5, #1, (R0)
53	61	01					BICB2 #16, (R0)
							EXTZV #11, #1, (R1), R3

60	01	07	53	FO	000BE	INSV	R3, #7, #1, (R0)	
53	61	01	0C	EF	000C3	EXTZV	#12, #1, (R1), R3	1963
60	01	09	53	FO	000C8	INSV	R3, #9, #1, (R0)	1952
			05	11	000CD	BRB	8\$	1977
			8F	AA	000CF	BICW2	#698, (R0)	1988
1C	60	02BA	01	7D	000D4	MOVQ	#1, 28(EVENT_ENTRY)	1994
	A6		A6	7C	000D8	CLRQ	36(EVENT_ENTRY)	1999
			A2	DO	000DB	MOVL	8(R2), NOUN_NODE	2004
	5A		A2	DO	000DF	MOVL	4(R2), ADVERB_NODE	2009
	52		03	12	000E3	BNEQ	10\$	2022
			0213	31	000E5	BRW	51\$	2024
			62	95	000E8	TSTB	(ADVERB_NODE)	2023
			07	12	000EA	BNEQ	11\$	2029
1C	A6	04	A2	DO	000EC	MOVL	4(ADVERB_NODE), 28(EVENT_ENTRY)	2030
			5D	11	000F1	BRB	16\$	2035
	01		62	91	000F3	CMPB	(ADVERB_NODE), #1	2037
			06	12	000F6	BNEQ	12\$	2038
17	A6		04	88	000F8	BISB2	#4, 23(EVENT_ENTRY)	2045
			52	11	000FC	BRB	16\$	2047
	02		62	91	000FE	CMPB	(ADVERB_NODE), #2	2048
			0A	12	00101	BNEQ	13\$	2049
17	A6		01	88	00103	BISB2	#1, 23(EVENT_ENTRY)	2055
19	A6		02	8A	00107	BICB2	#2, 25(EVENT_ENTRY)	2063
			0D	11	0010B	BRB	14\$	2064
	03		62	91	0010D	CMPB	(ADVERB_NODE), #3	2069
			0E	12	00110	BNEQ	15\$	2074
17	A6		01	8A	00112	BICB2	#1, 23(EVENT_ENTRY)	2075
19	A6		02	88	00116	BISB2	#2, 25(EVENT_ENTRY)	2076
19	A6		01	88	0011A	BISB2	#1, 25(EVENT_ENTRY)	2081
			62	11	0011E	BRB	18\$	2082
	04		62	91	00120	CMPB	(ADVERB_NODE), #4	2088
			2D	12	00123	BNEQ	17\$	2097
			04	DD	00125	PUSHL	#4	2098
00000000G	00		01	FB	00127	CALLS	#1, DBG\$GET MEMORY	2103
	58		50	DO	0012E	MOVL	R0, WHEN_ENTRY	2108
00000000'	EF		68	0E	00131	INSQUE	(WHEN_ENTRY), EVENTS\$WHEN_QUEUE	2109
20	A6		58	DO	00138	MOVL	WHEN_ENTRY, 32(EVENT_ENTRY)	2110
	53	04	A2	DO	0013C	MOVL	4(ADVERB_NODE), STRING	2115
	50		63	3C	00140	MOVZWL	(STRING), LENGTH	2116
01 A043	0D		90	00143	MOVB	#13, 1(LENGTH)[STRING]		2123
08 A8	01		DO	00148	MOVL	#1, 8(WHEN_ENTRY)		
0C A8	53		DO	0014C	MOVL	STRING, 12(WHEN_ENTRY)		
	30		11	00150	BRB	18\$		
	05		62	91	00152	CMPB	(ADVERB_NODE), #5	
			2D	12	00155	BNEQ	19\$	
			04	DD	00157	PUSHL	#4	
00000000G	00		01	FB	00159	CALLS	#1, DBG\$GET MEMORY	
	59		50	DO	00160	MOVL	R0, DO_ENTRY	
00000000'	EF		69	0E	00163	INSQUE	(DO_ENTRY), EVENTS\$DO_LIST_QUEUE	
24	A6	04	59	DO	0016A	MOVL	DO_ENTRY, 36(EVENT_ENTRY)	
	53		A2	DO	0016E	MOVL	4(ADVERB_NODE), STRING	
	50		63	3C	00172	MOVZWL	(STRING), LENGTH	
01 A043	0D		90	00175	MOVB	#13, 1(LENGTH)[STRING]		
08 A9	01		DO	0017A	MOVL	#1, 8(DO_ENTRY)		
0C A9	53		DO	0017E	MOVL	STRING, T2(DO_ENTRY)		
	64		11	00182	BRB	27\$		
	06		62	91	00184	CMPB	(ADVERB_NODE), #6	

0A		54	1F	00187	BLSSU	26\$		
		62	91	00189	CMPB	(ADVERB_NODE), #10		
		4F	1A	0018C	BGTRU	26\$		
	15	A6	94	0018E	CLRB	21(EVENT_ENTRY)		2130
06		62	91	00191	CMPB	(ADVERB_NODE), #6		2138
		04	12	00194	BNEQ	20\$		
		50	D4	00196	CLRL	R0		
		3D	11	00198	BRB	25\$		
07		62	91	0019A	20\$: CMPB	(ADVERB_NODE), #7		2140
		05	12	0019D	BNEQ	21\$		
50		01	D0	0019F	MOVL	#1, R0		
		33	11	001A2	BRB	25\$		
08		62	91	001A4	21\$: CMPB	(ADVERB_NODE), #8		2142
		05	12	001A7	BNEQ	22\$		
50		02	D0	001A9	MOVL	#2, R0		
		29	11	001AC	BRB	25\$		
09		62	91	001AE	22\$: CMPB	(ADVERB_NODE), #9		2144
		05	12	001B1	BNEQ	23\$		
50		03	D0	001B3	MOVL	#3, R0		
		1F	11	001B6	BRB	25\$		
0A		62	91	001B8	23\$: CMPB	(ADVERB_NODE), #10		2146
		05	12	001BB	BNEQ	24\$		
50		04	D0	001BD	MOVL	#4, R0		
		15	11	001C0	BRB	25\$		
	00000000'	EF	9F	001C2	24\$: PUSHAB	P.ABM		2149
		01	DD	001C8	PUSHL	#1		
	00028362	8F	DD	001CA	PUSHL	#164706		
00000000G	00	03	FB	001D0	CALLS	#3, LIB\$SIGNAL		
16	A6	50	90	001D7	25\$: MOVB	R0, 22(EVENT_ENTRY)		2136
		0B	11	001DB	BRB	27\$		2016
	0B	62	91	001DD	26\$: CMPB	(ADVERB_NODE), #11		2157
		09	12	001E0	BNEQ	28\$		
15	A6	8F	B0	001E2	MOVW	#3842, 21(EVENT_ENTRY)		2159
		0109	31	001E8	27\$: BRW	50\$		2016
	0A	62	91	001EB	28\$: CMPB	(ADVERB_NODE), #10		2167
		0D	13	001EE	BEQL	31\$		
12		62	91	001F0	CMPB	(ADVERB_NODE), #18		
		03	1E	001F3	BGEQU	30\$		
		0096	31	001F5	29\$: BRW	40\$		
16		62	91	001F8	30\$: CMPB	(ADVERB_NODE), #22		
		F8	1A	001FB	BGTRU	29\$		
15	A6	01	90	001FD	31\$: MOVB	#1, 21(EVENT_ENTRY)		2182
	12	62	91	00201	CMPB	(ADVERB_NODE), #18		2190
		05	12	00204	BNEQ	32\$		
50		05	D0	00206	MOVL	#5, R0		
		3D	11	00209	BRB	37\$		
13		62	91	0020B	32\$: CMPB	(ADVERB_NODE), #19		2192
		05	12	0020E	BNEQ	33\$		
50		06	D0	00210	MOVL	#6, R0		
		33	11	00213	BRB	37\$		
14		62	91	00215	33\$: CMPB	(ADVERB_NODE), #20		2194
		05	12	00218	BNEQ	34\$		
50		07	D0	0021A	MOVL	#7, R0		
		29	11	0021D	BRB	37\$		
16		62	91	0021F	34\$: CMPB	(ADVERB_NODE), #22		2196
		05	12	00222	BNEQ	35\$		
50		09	D0	00224	MOVL	#9, R0		

			1F 11 00227	BRB	37\$		
	15		62 91 00229	35\$: CMPB	(ADVERB_NODE), #21		2198
			05 12 0022C	BNEQ	36\$		
	50		08 D0 0022E	MOVL	#8, R0		
			15 11 00231	BRB	37\$		
		00000000'	EF 9F 00233	36\$: PUSHAB	P.ABN		2201
			01 DD 00239	PUSHL	#1		
		00028362	8F DD 0023B	PUSHL	#164706		
00000000G	00		03 FB 00241	CALLS	#3, LIB\$SIGNAL		
16	A6		50 90 00248	37\$: MOVB	R0, 22(EVENT_ENTRY)		2188
			50 D4 0024C	CLRL	R0		2209
	07	16	A6 91 0024E	CMPB	22(EVENT_ENTRY), #7		
			02 12 00252	BNEQ	38\$		
			50 D6 00254	INCL	R0		
18	A6	01	50 F0 00256	38\$: INSV	R0, #1, #1, 24(EVENT_ENTRY)		
			04 8A 0025C	BICB2	#4, 25(EVENT_ENTRY)		2214
	19	A6	50 D4 00260	CLRL	STATUS		2225
			6E D4 00262	CLRL	MODRST		2226
	07	16	A6 91 00264	CMPB	22(EVENT_ENTRY), #7		2227
			1B 12 00268	BNEQ	39\$		
			5E DD 0026A	PUSHL	SP		2234
		38	A6 9F 0026C	PUSHAB	56(EVENT_ENTRY)		
		34	A6 9F 0026F	PUSHAB	52(EVENT_ENTRY)		2233
		10	AE 9F 00272	PUSHAB	STMTNO		2229
		18	AE 9F 00275	PUSHAB	LINENO		
		00000000G	00 DD 00278	PUSHL	DBG\$RUNFRAME+64		2234
00000000G	00		06 FB 0027E	CALLS	#6, DBG\$PC TO_LINE_LOOKUP		
	6C		50 E8 00285	39\$: BLBS	STATUS, 50\$		2236
34	A6		01 7D 00288	MOVQ	#1, 52(EVENT_ENTRY)		2239
			66 11 0028C	BRB	50\$		2016
	0C		62 91 0028E	40\$: CMPB	(ADVERB_NODE), #12		2247
			06 12 00291	BNEQ	41\$		
	18	A6	20 88 00293	BISB2	#32, 24(EVENT_ENTRY)		2249
			09 11 00297	BRB	42\$		2250
	0D		62 91 00299	41\$: CMPB	(ADVERB_NODE), #13		2256
			0A 12 0029C	BNEQ	43\$		
	18	A6	20 8A 0029E	BICB2	#32, 24(EVENT_ENTRY)		2258
	18	A6	10 88 002A2	42\$: BISB2	#16, 24(EVENT_ENTRY)		2259
			4C 11 002A6	BRB	50\$		2016
	0E		62 91 002A8	43\$: CMPB	(ADVERB_NODE), #14		2265
			07 12 002AB	BNEQ	44\$		
	18	A6	80 8F 8A 002AD	BICB2	#128, 24(EVENT_ENTRY)		2267
			0A 11 002B2	BRB	45\$		2268
	0F		62 91 002B4	44\$: CMPB	(ADVERB_NODE), #15		2274
			0C 12 002B7	BNEQ	46\$		
	18	A6	80 8F 88 002B9	BISB2	#128, 24(EVENT_ENTRY)		2276
	18	A6	40 8F 88 002BE	45\$: BISB2	#64, 24(EVENT_ENTRY)		2277
			2F 11 002C3	BRB	50\$		2016
	10		62 91 002C5	46\$: CMPB	(ADVERB_NODE), #16		2283
			06 12 002C8	BNEQ	47\$		
	18	A6	08 8A 002CA	BICB2	#8, 24(EVENT_ENTRY)		2285
			09 11 002CE	BRB	48\$		2286
	11		62 91 002D0	47\$: CMPB	(ADVERB_NODE), #17		2292
			0A 12 002D3	BNEQ	49\$		
	18	A6	08 88 002D5	BISB2	#8, 24(EVENT_ENTRY)		2294
	18	A6	04 88 002D9	48\$: BISB2	#4, 24(EVENT_ENTRY)		2295
			15 11 002DD	BRB	50\$		2016

		52	2C	01F1	31	003B4	BRW	78\$		
		62	34	A6	9E	003B7	MOVAB	44	(EVENT_ENTRY), R2	2403
24	19	A6		05	D0	003BB	MOVL	ADDRESS	-(R2)	
			34	AE	E0	003BF	BBS	#5, 25	(EVENT_ENTRY), 60\$	2416
	00000000G	00		01	DD	003C4	PUSHL	ADDRESS		2425
		12		50	FB	003C7	CALLS	#1, DBG\$IS_IT_ENTRY		
		17	16	A7	E8	003CE	BLBS	R0, 59\$		
				0C	91	003D1	CMPB	22(R7), #23		2437
		20	16	A7	13	003D5	BEQL	59\$		
				06	91	003D7	CMPB	22(R7), #32		2438
		21	16	A7	13	003DB	BEQL	59\$		
				05	91	003DD	CMPB	22(R7), #33		2439
62	34	AE		02	12	003E1	BNEQ	60\$		
			34	02	C1	003E3	ADDL3	#2, ADDRESS, (R2)		2441
	00000000G	00		A6	9F	003E8	PUSHAB	52	(EVENT_ENTRY)	2449
		50		01	FB	003EB	CALLS	#1, DBG\$DATA_LENGTH		
7E		50	FD	07	C0	003F2	ADDL2	#7, R0		
				8F	78	003F5	ASHL	#-3, R0, -(SP)		2450
	00000000G	00		62	DD	003FA	PUSHL	(R2)		2448
		50	16	02	FB	003FC	CALLS	#2, DBG\$READ_ACCESS		
		03		A6	9A	00403	MOVZBL	22	(EVENT_ENTRY), R0	2456
				50	91	00407	CMPB	R0, #3		2458
		04		29	1F	0040A	BLSSU	61\$		
				50	91	0040C	CMPB	R0, #4		
		53		24	1A	0040F	BGTRU	61\$		
	10	AE		62	D0	00411	MOVL	(R2), EVENT_ADDRESS		2464
				63	9A	00414	MOVZBL	(EVENT_ADDRESS), OP		2467
			14	01	DD	00418	PUSHL	#1		
				AE	9F	0041A	PUSHAB	OP		
	00000000G	00		53	DD	0041D	PUSHL	EVENT_ADDRESS		
		72		03	FB	0041F	CALLS	#3, DBG\$WRITE_MEM		
				50	E8	00426	BLBS	R0, 63\$		
				53	DD	00429	PUSHL	EVENT_ADDRESS		2470
				01	DD	0042B	PUSHL	#1		
		00028220		8F	DD	0042D	PUSHL	#164384		
				7C	11	00433	BRB	67\$		
		02		50	91	00435	CMPB	R0, #2		2485
				61	12	00438	BNEQ	63\$		
				7E	D4	0043A	CLRL	-(SP)		2506
			24	AE	9F	0043C	PUSHAB	DUMMY		
			1C	AE	9F	0043F	PUSHAB	PRIMARY		
			28	A6	DD	00442	PUSHL	40	(EVENT_ENTRY)	
	00000000G	00		04	FB	00445	CALLS	#4, DBG\$NCOPY_DESC		
		53	14	AE	D0	0044C	MOVL	PRIMARY, R3		2509
		0D	04	A3	E9	00450	BLBC	4(R3), 62\$		
			00028A4A	6F	DD	00454	PUSHL	#166474		2511
	00000000G	00		01	FB	0045A	CALLS	#1, LIB\$SIGNAL		
			18	AE	9F	00461	PUSHAB	VALUE		2518
		7E	83	8F	9A	00464	MOVZBL	#131, -(SP)		
				53	DD	00468	PUSHL	R3		
	00000000G	00		03	FB	0046A	CALLS	#3, DBG\$PRIM_TO_VAL		
			18	AE	9F	00471	PUSHAB	VALUE		2521
		7E	7A	8F	9A	00474	MOVZBL	#122, -(SP)		
			20	AE	DD	00478	PUSHL	VALUE		
	00000000G	00		03	FB	0047B	CALLS	#3, DBG\$PRIM TO VAL		
0000007A	8F		18	10	ED	00482	CMPZV	#16, #8, @VALUE, #122		2526
				10	13	0048C	BEQL	64\$		

00000000G	00	00028A52	8F	DD	0048E	PUSHL	#166482	2528
			01	FB	00494	CALLS	#1, LIB\$SIGNAL	
40000000	8F		013D	31	0049B	BRW	81\$	
			62	D1	0049E	63\$:		
			14	1F	004A5	64\$:		2541
			62	DD	004A7	65\$:		
			01	DD	004A9	66\$:		2545
			8F	DD	004AB	PUSHL	#1	2544
00000000G	00	00028230	03	FB	004B1	PUSHL	#164400	
			00ED	31	004B8	CALLS	#3, LIB\$SIGNAL	
		20	AE	9F	004BB	BRW	78\$	2546
			62	DD	004BE	68\$:		2555
00000000G	00		02	FB	004C0	PUSHAB	DUMMY	
	09		50	E9	004C7	PUSHL	(R2)	
		00028CA8	8F	DD	004CA	CALLS	#2, DBG\$MAP_TO_REG_ADDR	
			00CE	31	004D0	BLBC	R0, 69\$	
50	00000000G		00	9E	004D3	PUSHL	#167080	2558
50			62	D1	004DA	BRW	77\$	
			0C	1F	004DD	MOVAB	DBG\$RUNFRAME+4, R0	2562
50	00000000G		00	9E	004DF	CMPL	(R2), R0	
50			62	D1	004E6	BLSSU	70\$	2565
			BC	1F	004E9	MOVAB	DBG\$RUNFRAME+72, R0	
		20	AE	9F	004EB	CMPL	(R2), R0	
		20	AE	9F	004EE	BLSSU	65\$	2579
		20	AE	DD	004F1	PUSHAB	DUMMY	
00000000G	00		03	FB	004F4	PUSHAB	SYMID_LIST	
		1C	AE	DD	004FB	PUSHL	VALUE	
00000000G	00		01	FB	004FE	CALLS	#3, DBG\$NGET_SYMID	2580
		20	AE	9F	00505	PUSHL	SYMID_LIST	
		30	A6	9F	00508	CALLS	#1, DBG\$STA_LOCK_SYMID	
		20	AE	DD	0050B	PUSHAB	DUMMY	2586
00000000G	00		03	FB	0050E	PUSHAB	48(EVENT_ENTRY)	
17	A6		08	88	00515	PUSHL	VALUE	
		08	AC	DD	00519	CALLS	#3, DBG\$NCOPY_DESC	
		28	AE	9F	0051C	BISB2	#8, 23(EVENT_ENTRY)	2593
		28	A6	DD	0051F	PUSHL	MESSAGE_VECT	2600
00000000G	00		03	FB	00522	PUSHAB	PAGE_LIST	2599
	03		50	E8	00529	PUSHL	40(EVENT_ENTRY)	
		FF	78	31	0052C	CALLS	#3, DBG\$NGET_PAGES	
54	24		AE	D0	0052F	BLBS	R0, 71\$	
52	00000000'		EF	D0	00533	BRW	65\$	
			55	D4	0053A	MOVL	PAGE_LIST, R4	2624
50	00000000'		EF	9E	0053C	71\$:		2619
50			52	D1	00543	72\$:		2620
			14	13	00546	CLRL	PAGE_FOUND	2620
04	A4	08	A2	D1	00548	MOVAB	EVENT\$PAGE_QUEUE, R0	2621
			08	12	0054D	CMPL	PAGE_ENTRY, R0	
		0C	A2	B6	0054F	BEQL	75\$	
			01	D0	00552	CMPL	8(PAGE_ENTRY), 4(R4)	2624
			05	11	00555	BNEQ	74\$	
52			62	D0	00557	INCW	12(PAGE_ENTRY)	2634
			E0	11	0055A	MOVL	#1, PAGE_FOUND	2635
6D			55	E8	0055C	BRB	75\$	2632
53	24		AE	D0	0055F	74\$:		2643
2C	04		A3	D0	00563	BRB	73\$	2621
30	04		A3	D0	00568	BLBS	PAGE_FOUND, 80\$	2651
	28		AE	9F	0056D	MOVL	PAGE_LIST, R3	2658
						MOVL	4(R3), ADDRESS	
						MOVL	4(R3), ADDRESS+4	2659
						PUSHAB	PROTECT	2663

			0F DD 00570	PUSHL #15	
			7E 7C 00572	CLRQ -(SP)	
		3C	AE 9F 00574	PUSHAB ADDRESS	
00000000G	00		05 FB 00577	CALLS #5, SYS\$SETPRT	
	06		50 E8 0057E	BLBS R0, 76\$	
		04	A3 DD 00581	PUSHL 4(R3)	2666
		FF22	31 00584	BRW 66\$	
			7E D4 00587 76\$:	CLRL -(SP)	2671
		2C	AE DD 00589	PUSHL PROTECT	
			7E 7C 0058C	CLRQ -(SP)	
		3C	AE 9F 0058E	PUSHAB ADDRESS	
00000000G	00		05 FB 00591	CALLS #5, SYS\$SETPRT	
	11		50 E8 00598	BLBS R0, 79\$	
		000284C4	8F DD 0059B	PUSHL #165060	2674
00000000G	00		01 FB 005A1 77\$:	CALLS #1, LIB\$SIGNAL	
	50		04 D0 005A8 78\$:	MOVL #4, R0	2675
			04 005AB	RET	
			04 DD 005AC 79\$:	PUSHL #4	2677
00000000G	00		01 FB 005AE	CALLS #1, DBG\$GET_MEMORY	
	52		50 D0 005B5	MOVL R0, PAGE_ENTRY	
000000000'	EF		62 0E 005B8	INSQUE (PAGE_ENTRY), EVENT\$PAGE_QUEUE	2678
	50	24	AE D0 005BF	MOVL PAGE_LIST, R0	2680
	A2	04	A0 D0 005C3	MOVL 4(R0), 8(PAGE_ENTRY)	
	OC		01 B0 005C8	MOVW #1, 12(PAGE_ENTRY)	2681
	24	24	BE D0 005CC 80\$:	MOVL @PAGE_LIST, PAGE_LIST	2683
	54	24	AE D0 005D1	MOVL PAGE_LIST, R4	2685
	03		54 E9 005D5	BLBC R4, 81\$	
		FF58	31 005D8	BRW 72\$	
	5A	08	AA D0 005DB 81\$:	MOVL 8(NOUN_NODE), NOUN_NODE	2693
			52 13 005DF	BEQL 87\$	2700
			10 DD 005E1	PUSHL #16	2709
00000000G	00		01 FB 005E3	CALLS #1, DBG\$GET_MEMORY	
	56		50 D0 005EA	MOVL R0, EVENT_ENTRY	
66	3C	0040	8F 28 005ED	MOVC3 #64, @EVENT_QUEUE, (EVENT_ENTRY)	2713
	3C		66 0E 005F4	INSQUE (EVENT_ENTRY), EVENT_QUEUE	2719
		20	A6 D5 005F8	TSTL 32(EVENT_ENTRY)	2725
			03 13 005FB	BEQL 82\$	
		08	A8 D6 005FD	INCL 8(WHEN_ENTRY)	2728
		24	A6 D5 00600 82\$:	TSTL 36(EVENT_ENTRY)	2734
			03 13 00603	BEQL 83\$	
		08	A9 D6 00605	INCL 8(DO_ENTRY)	2737
		FD1B	31 00608 83\$:	BRW 54\$	2700
	01		50 91 0060B 84\$:	CMPB R0, #1	2751
			03 13 0060E	BEQL 85\$	
		00A8	31 00610	BRW 99\$	
	50	16	A6 9A 00613 85\$:	MOVZBL 22(EVENT_ENTRY), R0	2752
	05		50 91 00617	CMPB R0, #5	2754
			0A 12 0061A	BNEQ 86\$	
28	A6	00000000'	EF 9E 0061C	MOVAB DBG\$OPCODES_CALL, 40(EVENT_ENTRY)	2755
			0D 11 00624	BRB 87\$	
	06		50 91 00626 86\$:	CMPB R0, #6	2756
			0A 12 00629	BNEQ 88\$	
28	A6	00000000'	EF 9E 0062B	MOVAB DBG\$OPCODES_BRANCH, 40(EVENT_ENTRY)	2757
			7C 11 00633 87\$:	BRB 97\$	
	08		50 91 00635 88\$:	CMPB R0, #8	2758
			74 13 00638	BEQL 96\$	
	09		50 91 0063A	CMPB R0, #9	2760

PC	Op	OpC	OpD	OpI	OpR	OpS	OpT	OpV	OpW	OpX	OpY	OpZ	OpAA	OpAB	OpAC	OpAD	OpAE	OpAF	OpAG	OpAH	OpAI	OpAJ	OpAK	OpAL	OpAM	OpAN	OpAO	OpAP	OpAQ	OpAR	OpAS	OpAT	OpAU	OpAV	OpAW	OpAX	OpAY	OpAZ	OpBA	OpBB	OpBC	OpBD	OpBE	OpBF	OpBG	OpBH	OpBI	OpBJ	OpBK	OpBL	OpBM	OpBN	OpBO	OpBP	OpBQ	OpBR	OpBS	OpBT	OpBU	OpBV	OpBW	OpBX	OpBY	OpBZ	OpCA	OpCB	OpCC	OpCD	OpCE	OpCF	OpCG	OpCH	OpCI	OpCJ	OpCK	OpCL	OpCM	OpCN	OpCO	OpCP	OpCQ	OpCR	OpCS	OpCT	OpCU	OpCV	OpCW	OpCX	OpCY	OpCZ	OpDA	OpDB	OpDC	OpDD	OpDE	OpDF	OpDG	OpDH	OpDI	OpDJ	OpDK	OpDL	OpDM	OpDN	OpDO	OpDP	OpDQ	OpDR	OpDS	OpDT	OpDU	OpDV	OpDW	OpDX	OpDY	OpDZ	OpEA	OpEB	OpEC	OpED	OpEE	OpEF	OpEG	OpEH	OpEI	OpEJ	OpEK	OpEL	OpEM	OpEN	OpEO	OpEP	OpEQ	OpER	OpES	OpET	OpEU	OpEV	OpEW	OpEX	OpEY	OpEZ	OpFA	OpFB	OpFC	OpFD	OpFE	OpFF	OpFG	OpFH	OpFI	OpFJ	OpFK	OpFL	OpFM	OpFN	OpFO	OpFP	OpFQ	OpFR	OpFS	OpFT	OpFU	OpFV	OpFW	OpFX	OpFY	OpFZ	OpGA	OpGB	OpGC	OpGD	OpGE	OpGF	OpGG	OpGH	OpGI	OpGJ	OpGK	OpGL	OpGM	OpGN	OpGO	OpGP	OpGQ	OpGR	OpGS	OpGT	OpGU	OpGV	OpGW	OpGX	OpGY	OpGZ	OpHA	OpHB	OpHC	OpHD	OpHE	OpHF	OpHG	OpHH	OpHI	OpHJ	OpHK	OpHL	OpHM	OpHN	OpHO	OpHP	OpHQ	OpHR	OpHS	OpHT	OpHU	OpHV	OpHW	OpHX	OpHY	OpHZ	OpIA	OpIB	OpIC	OpID	OpIE	OpIF	OpIG	OpIH	OpII	OpIJ	OpIK	OpIL	OpIM	OpIN	OpIO	OpIP	OpIQ	OpIR	OpIS	OpIT	OpIU	OpIV	OpIW	OpIX	OpIY	OpIZ	OpJA	OpJB	OpJC	OpJD	OpJE	OpJF	OpJG	OpJH	OpJI	OpJJ	OpJK	OpJL	OpJM	OpJN	OpJO	OpJP	OpJQ	OpJR	OpJS	OpJT	OpJU	OpJV	OpJW	OpJX	OpJY	OpJZ	OpKA	OpKB	OpKC	OpKD	OpKE	OpKF	OpKG	OpKH	OpKI	OpKJ	OpKK	OpKL	OpKM	OpKN	OpKO	OpKP	OpKQ	OpKR	OpKS	OpKT	OpKU	OpKV	OpKW	OpKX	OpKY	OpKZ	OpLA	OpLB	OpLC	OpLD	OpLE	OpLF	OpLG	OpLH	OpLI	OpLJ	OpLK	OpLL	OpLM	OpLN	OpLO	OpLP	OpLQ	OpLR	OpLS	OpLT	OpLU	OpLV	OpLW	OpLX	OpLY	OpLZ	OpMA	OpMB	OpMC	OpMD	OpME	OpMF	OpMG	OpMH	OpMI	OpMJ	OpMK	OpML	OpMM	OpMN	OpMO	OpMP	OpMQ	OpMR	OpMS	OpMT	OpMU	OpMV	OpMW	OpMX	OpMY	OpMZ	OpNA	OpNB	OpNC	OpND	OpNE	OpNF	OpNG	OpNH	OpNI	OpNJ	OpNK	OpNL	OpNM	OpNN	OpNO	OpNP	OpNQ	OpNR	OpNS	OpNT	OpNU	OpNV	OpNW	OpNX	OpNY	OpNZ	OpOA	OpOB	OpOC	OpOD	OpOE	OpOF	OpOG	OpOH	OpOI	OpOJ	OpOK	OpOL	OpOM	OpON	OpOO	OpOP	OpOQ	OpOR	OpOS	OpOT	OpOU	OpOV	OpOW	OpOX	OpOY	OpOZ	OpPA	OpPB	OpPC	OpPD	OpPE	OpPF	OpPG	OpPH	OpPI	OpPJ	OpPK	OpPL	OpPM	OpPN	OpPO	OpPP	OpPQ	OpPR	OpPS	OpPT	OpPU	OpPV	OpPW	OpPX	OpPY	OpPZ	OpQA	OpQB	OpQC	OpQD	OpQE	OpQF	OpQG	OpQH	OpQI	OpQJ	OpQK	OpQL	OpQM	OpQN	OpQO	OpQP	OpQQ	OpQR	OpQS	OpQT	OpQU	OpQV	OpQW	OpQX	OpQY	OpQZ	OpRA	OpRB	OpRC	OpRD	OpRE	OpRF	OpRG	OpRH	OpRI	OpRJ	OpRK	OpRL	OpRM	OpRN	OpRO	OpRP	OpRQ	OpRR	OpRS	OpRT	OpRU	OpRV	OpRW	OpRX	OpRY	OpRZ	OpSA	OpSB	OpSC	OpSD	OpSE	OpSF	OpSG	OpSH	OpSI	OpSJ	OpSK	OpSL	OpSM	OpSN	OpSO	OpSP	OpSQ	OpSR	OpSS	OpST	OpSU	OpSV	OpSW
----	----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------

				71	11	00725	105\$:	BRB	109\$		
				02	DD	00727	106\$:	PUSHL	#2		2887
				01	FB	00729		CALLS	#1, DBG\$SET_STP_LVL		
				A6	D0	00730		MOVL	28(EVENT_ENTRY), DBG\$GL_STEP_NUM		2892
				5E	13	00738		BEQL	109\$		2898
				66	0F	0073A		REMQUE	(EVENT_ENTRY), EVENT_ENTRY		2907
				66	0E	0073D		INSQUE	(EVENT_ENTRY), EVENT\$CMD_QUEUE		2908
				56	D0	00744		MOVL	EVENT_ENTRY, 8(EVENT_ENTRY)		2914
				56	D0	00748		MOVL	EVENT_ENTRY, 12(EVENT_ENTRY)		2915
				A6	91	0074C		CMPB	22(EVENT_ENTRY), #4		2923
				40	12	00750		BNEQ	108\$		
				00	D0	00752		MOVL	DBG\$RUNFRAME+64, R0		2933
				00	D0	00759		MOVL	DBG\$RUNFRAME+56, R1		2934
				50	7D	00760		MOVQ	R0, 32(EVENT_ENTRY)		2933
				00	0C	00764		PROBER	#0, #1, (R0)		2935
				16	13	00768		BEQL	107\$		
				60	90	0076A		MOVB	(R0), INSTRUCTION		2938
				52	91	0076D		CMPB	INSTRUCTION, #4		2939
				0E	12	00770		BNEQ	107\$		
				51	D0	00772		MOVL	R1, STACK_FRAME		2942
				00	0C	00775		PROBER	#0, #20, (STACK_FRAME)		2943
				05	13	00779		BEQL	107\$		
				A0	D0	0077B		MOVL	12(STACK_FRAME), 36(EVENT_ENTRY)		2946
				05	DD	00780	107\$:	PUSHL	#5		2952
				01	FB	00782		CALLS	#1, DBG\$GET_MEMORY		
				50	D0	00789		MOVL	CALL_FRAME, -60(EVENT_ENTRY)		2953
				14	28	0078D		MOVC3	#20, -36(EVENT_ENTRY), (CALL_FRAME)		2954
				00	94	00792	108\$:	CLRB	DBG\$GB_TAKE_CMD		2960
				00	D0	00798	109\$:	MOVL	DBG\$GB_STP_PTR, R7		2972
				A6	90	0079F		MOVB	22(EVENT_ENTRY), (R7)		2973
				A6	9E	007A3		MOVAB	24(EVENT_ENTRY), R0		2975
				01	EF	007A7		EXTZV	#1, #1, (R0), R1		
				51	F0	007AC		INSV	R1, #0, #1, (R7)		
				03	EF	007B2		EXTZV	#3, #1, (R0), R1		2977
				51	F0	007B7		INSV	R1, #9, #1, (R7)		
				05	EF	007BC		EXTZV	#5, #1, (R0), R1		2979
				51	F0	007C1		INSV	R1, #10, #1, (R7)		
				07	EF	007C6		EXTZV	#7, #1, (R0), R1		2981
				51	F0	007CB		INSV	R1, #11, #1, (R7)		
				09	EF	007D0		EXTZV	#9, #1, (R0), R1		2983
				51	F0	007D5		INSV	R1, #12, #1, (R7)		
				08	ED	007DA		CMPZV	#8, #8, @VERB_NODE, #11		2989
				21	12	007E0		BNEQ	111\$		
				A6	91	007E2		CMPB	22(EVENT_ENTRY), #9		2992
				18	12	007E6		BNEQ	110\$		
				8F	28	007E8		MOVC3	#64, DBG\$OPCODES_USER, -		2995
				EF	9E	007F6		MOVAB	DBG\$OPCODES_STEP_USER		
				03	11	007FE		BRB	111\$		2996
				A7	D4	00800	110\$:	CLRL	4(R7)		2992
				31	00803		111\$:	BRW	128\$		3000
				AE	9E	00806	112\$:	MOVAB	EVENT_QUEUE, R0		2989
				AE	D1	0080A		CMPB	EVENT_QUEUE, R0		3021
				F3	13	0080E		BEQL	111\$		
				AE	D0	00810		MOVL	EVENT_QUEUE, EVENT_ENTRY		3028
				EF	D0	00814		MOVL	EVENT\$CMD_QUEUE, QUEUE_ENTRY		3033
				62	D0	0081B	113\$:	MOVL	(QUEUE_ENTRY), NEXTQ_ENTRY		3034

50	00000000'	EF	9E	0081E	MOVAB	EVENT\$CMD QUEUE, R0	3039
50		52	D1	00825	CMPL	QUEUE_ENTRY, R0	
		03	12	00828	BNEQ	114\$	
		0099	31	0082A	BRW	127\$	
53	14	A2	9E	0082D	114\$: MOVAB	20(QUEUE_ENTRY), R3	3046
		63	D5	00831	TSTL	(R3)	
		74	19	00833	BLSS	124\$	
50	15	A6	9A	00835	MOVZBL	21(EVENT_ENTRY), R0	3054
50	01	A3	91	00839	CMPB	1(R3), R0	3055
		6A	12	0083D	BNEQ	124\$	
		50	D5	0083F	TSTL	R0	3071
		35	12	00841	BNEQ	118\$	
50	16	A6	9A	00843	MOVZBL	22(EVENT_ENTRY), CODE_1	3081
51	02	A3	9A	00847	MOVZBL	2(R3), CODE_2	3082
03		50	D1	0084B	CMPL	CODE_1, #3	3083
		0F	13	0084E	BEQL	115\$	
04		50	D1	00850	CMPL	CODE_1, #4	3084
		0A	13	00853	BEQL	115\$	
03		51	D1	00855	CMPL	CODE_2, #3	3085
		05	13	00858	BEQL	115\$	
04		51	D1	0085A	CMPL	CODE_2, #4	3086
		09	12	0085D	BNEQ	117\$	
2C	A2	2C	A6	D1	0085F	115\$: CMPL	44(EVENT_ENTRY), 44(QUEUE_ENTRY)
			5A	12	00864	116\$: BNEQ	126\$
			3C	11	00866	BRB	123\$
		34	A2	9F	00868	117\$: PUSHAB	52(QUEUE_ENTRY)
		34	A6	9F	0086B	PUSHAB	52(EVENT_ENTRY)
0000V	CF		02	FB	0086E	CALLS	#2, COMPARE_VMSDESC
	4A		50	E9	00873	BLBC	R0, 126\$
			2C	11	00876	BRB	123\$
	01		50	91	00878	118\$: CMPB	R0, #1
			22	12	0087B	BNEQ	122\$
0011	04	05	16	A6	8F	0087D	CASEB
	000A	000A	000A		00882	119\$: .WORD	120\$-119\$,-
			0011		0088A		120\$-119\$,-
							120\$-119\$,-
							121\$-119\$,-
							121\$-119\$,-
							121\$-119\$
							22(EVENT_ENTRY), 2(R3)
02	A3	16	A6	91	0088C	120\$: CMPB	
			D1	11	00891	BRB	116\$
			A3	91	00893	121\$: CMPB	2(R3), #8
08	02		0B	13	00897	BEQL	123\$
			A3	91	00899	CMPB	2(R3), #9
09	02		C5	11	0089D	BRB	116\$
			50	91	0089F	122\$: CMPB	R0, #2
			07	12	008A2	BNEQ	125\$
03	A3	80	8F	88	008A4	123\$: BISB2	#128, 3(R3)
			15	11	008A9	124\$: BRB	126\$
		00000000'	EF	9F	008AB	125\$: PUSHAB	P.ABS
			01	DD	008B1	PUSHL	#1
		00028362	8F	DD	008B3	PUSHL	#164706
00000000G	00		03	FB	008B9	CALLS	#3, LIB\$SIGNAL
	52		54	DD	008C0	126\$: MOVL	NEXTQ_ENTRY, QUEUE_ENTRY
			FF	31	008C3	BRW	113\$
	56		66	0F	008C6	127\$: REMQUE	(EVENT_ENTRY), EVENT_ENTRY
00000000'	FF		66	0E	008C9	INSQUE	(EVENT_ENTRY), @EVENT\$CMD QUEUE+4
08	A6		56	DD	008D0	MOVL	EVENT_ENTRY, 8(EVENT_ENTRY)

DBGEVENT
V04-000

G 12
16-Sep-1984 00:59:10
14-Sep-1984 12:16:53

VAX-11 Bliss-32 V4.0-742
[DEBUG.SRC]DBGEVENT.B32;1

Page 89
(9)

0C	A6	56	D0	008D4	MOVL	EVENT_ENTRY, 12(EVENT_ENTRY)	:	3161		
		FF2B	31	008D8	BRW	112\$:	3021		
	56	3C	AE	D0	008DB	128\$:	MOVL	EVENT_QUEUE, EVENT_ENTRY	:	3170
	50	3C	AE	9E	008DF	129\$:	MOVAB	EVENT_QUEUE, R0	:	3171
	50		56	D1	008E3		CMPL	EVENT_ENTRY, R0	:	
			0F	13	008E6		BEQL	130\$:	
	52		66	D0	008E8		MOVL	(EVENT_ENTRY), NEW_EVENT_ENTRY	:	3175
			56	DD	008EB		PUSHL	EVENT_ENTRY	:	3176
0000V	CF		01	FB	008ED		CALLS	#1, DELETE_EVENT_ENTRY	:	
	56		52	D0	008F2		MOVL	NEW_EVENT_ENTRY, EVENT_ENTRY	:	3177
			E8	11	008F5		BRB	129\$:	3171
	50		01	D0	008F7	130\$:	MOVL	#1, R0	:	3183
			04	008FA		RET			:	3185

; Routine Size: 2299 bytes, Routine Base: DBG\$CODE + 0AEB

```
3062 3186 1 GLOBAL ROUTINE DBG$EVENT_SHOW_CANCEL_SYNTAX(INPUT_DESC,  
3063 3187 1 VERB_NODE, MESSAGE_VECT) =  
3064 3188 1  
3065 3189 1 FUNCTION  
3066 3190 1 This routine parses a show or cancel break, trace, or watch point  
3067 3191 1 command,  
3068 3192 1 after the '[SHOW|CANCEL] BREAK', '[SHOW|CANCEL] TRACE', or  
3069 3193 1 '[SHOW|CANCEL] WATCH' has been parsed. This routine is called  
3070 3194 1 from DBGNSET.  
3071 3195 1  
3072 3196 1 The following syntax is accepted:  
3073 3197 1  
3074 3198 1 <event>  
3075 3199 1  
3076 3200 1 If the event is a watchpoint, <event> is:  
3077 3201 1  
3078 3202 1 /ALL ;  
3079 3203 1 [ <address> [ , <address> ]*]  
3080 3204 1  
3081 3205 1 otherwise, <event> is:  
3082 3206 1  
3083 3207 1 /EXCEPTION ;  
3084 3208 1 /CALL ;  
3085 3209 1 /RETURN ;  
3086 3210 1 /BRANCH ;  
3087 3211 1 /LINE ;  
3088 3212 1 /INSTRUCTION [ = ( <mnemonic> [ , <mnemonic> ] ) ]  
3089 3213 1 /ALL ;  
3090 3214 1 [ <address> [ , <address> ]*]  
3091 3215 1  
3092 3216 1 INPUTS  
3093 3217 1 INPUT_DESC: The current command line input descriptor.  
3094 3218 1 VERB_NODE: A pointer to a DEBUG verb node, with the COMPOSITE  
3095 3219 1 attribute set to:  
3096 3220 1  
3097 3221 1 EVENT$K_SHOW_BREAK, EVENT$K_SHOW_TRACE,  
3098 3222 1 EVENT$K_SHOW_WATCH,  
3099 3223 1 EVENT$K_CANCEL_BREAK, EVENT$K_CANCEL_TRACE,  
3100 3224 1 EVENT$K_CANCEL_WATCH.  
3101 3225 1  
3102 3226 1 MESSAGE_VECT: The address of a longword to contain the address  
3103 3227 1 of a message argument vector.  
3104 3228 1  
3105 3229 1 OUTPUTS  
3106 3230 1 The parse tree headed by VERB_NODE is created to represent the  
3107 3231 1 command parsed. Errors cause this routine to abort.  
3108 3232 1  
3109 3233 1 BEGIN  
3110 3234 1  
3111 3235 1 MAP  
3112 3236 1 VERB_NODE : REF DBG$VERB_NODE; ! Verb node pointer  
3113 3237 1  
3114 3238 1 LOCAL  
3115 3239 1 STATUS, ! Returned status  
3116 3240 1 ADVERB_NODE : REF DBG$ADVERB_NODE; ! Adverb node pointer  
3117 3241 1 ADVERB_LINK : REF DBG$ADVERB_NODE; ! Adverb node link  
3118 3242 1 NOUN_NODE : REF DBG$NOUN_NODE; ! Noun node pointer
```



```

: 3119
: 3120
: 3121
: 3122
: 3123
: 3124
: 3125
: 3126
: 3127
: 3128
: 3129
: 3130
: 3131
: 3132
: 3133
: 3134
: 3135
: 3136
: 3137
: 3138
: 3139
: 3140
: 3141
: 3142
: 3143
: 3144
: 3145
: 3146
: 3147
: 3148
: 3149
: 3150
: 3151
: 3152
: 3153
: 3154
: 3155
: 3156
: 3157
: 3158
: 3159
: 3160
: 3161
: 3162
: 3163
: 3164
: 3165
: 3166
: 3167
: 3168
: 3169
: 3170
: 3171
: 3172
: 3173
: 3174
: 3175

```

```

NOUN_LINK : REF DBG$NOUN_NODE; ! Noun node link

BIND

! Define the character strings to parse.
DBG$CS_EXCEPTION = SAC ('EXCEPTION'),
DBG$CS_INSTRUCTION = SAC ('INSTRUCTION'),
DBG$CS_CALL = SAC ('CALL'),
DBG$CS_RETURN = SAC ('RETURN'),
DBG$CS_BRANCH = SAC ('BRANCH'),
DBG$CS_LINE = SAC ('LINE'),
DBG$CS_ALL = SAC ('ALL'),
DBG$CS_COMMA = SAC (','),
DBG$CS_SLASH = SAC ('/'),
DBG$CS_CR = UPLIT BYTE (1, DBG$K_CAR_RETURN);

! Initialize status to success, in case we don't need to
! parse any addresses.
STATUS = ST$K_SUCCESS;

! Initialize the verb's adverb and noun pointers to null.
VERB_NODE [DBG$L_VERB_ADVERB_PTR] = 0;
VERB_NODE [DBG$L_VERB_OBJECT_PTR] = 0;

! Initialize the adverb and noun links.
ADVERB_LINK = VERB_NODE [DBG$L_VERB_ADVERB_PTR];
NOUN_LINK = VERB_NODE [DBG$L_VERB_OBJECT_PTR];

! Case on the verb composite - the command type is [SHOW:CANCEL] BREAK,
! [SHOW:CANCEL] TRACE, or [SHOW:CANCEL] WATCH.
SELECTONE .VERB_NODE [DBG$B_VERB_COMPOSITE] OF
SET

! [SHOW:CANCEL] BREAK and [SHOW:CANCEL] TRACE come here,
! to determine the event type.
[EVENT$K_SHOW_BREAK, EVENT$K_CANCEL_BREAK,
EVENT$K_SHOW_TRACE, EVENT$K_CANCEL_TRACE]:
BEGIN

! Parse the command according to the following syntax:
! /EXCEPTION
! /CALL
! /RETURN

```

```
.. 3176 3300 3
... 3177 3301 3
... 3178 3302 3
... 3179 3303 3
... 3180 3304 3
... 3181 3305 3
... 3182 3306 3
... 3183 3307 3
... 3184 3308 4
... 3185 3309 3
... 3186 3310 4
... 3187 3311 4
... 3188 3312 4
... 3189 3313 4
... 3190 3314 4
... 3191 3315 4
... 3192 3316 4
... 3193 3317 4
... 3194 3318 4
... 3195 3319 4
... 3196 3320 4
... 3197 3321 4
... 3198 3322 4
... 3199 3323 4
... 3200 3324 4
... 3201 3325 4
... 3202 3326 4
... 3203 3327 4
... 3204 3328 4
... 3205 3329 4
... 3206 3330 4
... 3207 3331 4
... 3208 3332 4
... 3209 3333 4
... 3210 3334 4
... 3211 3335 4
... 3212 3336 4
... 3213 3337 4
... 3214 3338 4
... 3215 3339 4
... 3216 3340 4
... 3217 3341 4
... 3218 3342 4
... 3219 3343 4
... 3220 3344 4
... 3221 3345 4
... 3222 3346 4
... 3223 3347 4
... 3224 3348 4
... 3225 3349 4
... 3226 3350 4
... 3227 3351 4
... 3228 3352 4
... 3229 3353 4
... 3230 3354 4
... 3231 3355 4
.. 3232 3356 4
```

```
... /BRANCH
... /LINE
... /INSTRUCTION
... /ALL
... [ <address> [ , <address> ]*]
... First, look for a '/'...
... IF MATCH (DBG$CS_SLASH)
... THEN
... BEGIN
...
... ! Get an node for the qualifier.
... !
... GET_ADVERB_NODE;
...
... ! Parse the qualifier.
... !
... SELECTONE TRUE OF
... SET
...
... ! Parse /EXCEPTION.
... !
... [MATCH (DBG$CS_EXCEPTION)]:
... ADVERB_NODE [DBG$B_ADVERB_LITERAL] =
... ADVERB_EXCEPTION;
...
... ! Parse /CALL.
... !
... [MATCH (DBG$CS_CALL)]:
... ADVERB_NODE [DBG$B_ADVERB_LITERAL] =
... ADVERB_CALL;
...
... ! Parse /BRANCH.
... !
... [MATCH (DBG$CS_BRANCH)]:
... ADVERB_NODE [DBG$B_ADVERB_LITERAL] =
... ADVERB_BRANCH;
...
... ! Parse /LINE.
... !
... [MATCH (DBG$CS_LINE)]:
... ADVERB_NODE [DBG$B_ADVERB_LITERAL] =
... ADVERB_LINE;
...
... ! Parse /INSTRUCTION
... !
... [MATCH (DBG$CS_INSTRUCTION)]:
... ADVERB_NODE [DBG$B_ADVERB_LITERAL] =
... ADVERB_INSTRUCTION;
```



```
3233 3357 4
3234 3358 4
3235 3359 4
3236 3360 4
3237 3361 4
3238 3362 4
3239 3363 4
3240 3364 4
3241 3365 4
3242 3366 4
3243 3367 4
3244 3368 4
3245 3369 4
3246 3370 4
3247 3371 4
3248 3372 4
3249 3373 4
3250 3374 4
3251 3375 4
3252 3376 4
3253 3377 4
3254 3378 4
3255 3379 4
3256 3380 4
3257 3381 4
3258 3382 4
3259 3383 4
3260 3384 4
3261 3385 5
3262 3386 4
3263 3387 4
3264 3388 5
3265 3389 5
3266 3390 5
3267 3391 5
3268 3392 5
3269 3393 5
3270 3394 5
3271 3395 5
3272 3396 5
3273 3397 5
3274 3398 5
3275 3399 5
3276 3400 5
3277 3401 5
3278 3402 5
3279 3403 5
3280 3404 5
3281 3405 5
3282 3406 5
3283 3407 5
3284 3408 5
3285 3409 5
3286 3410 5
3287 3411 5
3288 3412 5
3289 3413 5

! Parse /ALL
[ MATCH (DBG$CS_ALL) ]:
  ADVERB_NODE [DBG$B_ADVERB_LITERAL] =
    ADVERB_ALL;

! Otherwise, we've got a syntax error....
[ OTHERWISE ]:
  SYNTAX_ERROR;
TES
END
ELSE

! Parse the defaulting <address> [ , <address> ]*.
BEGIN

! If we're not standing on a <cr>, get the
! <address> [ , <address> ], as long as
! an <address> is terminated with a ','.
IF (NOT MATCH (DBG$CS_CR))
THEN
DO
BEGIN
! Get a noun node for an <address>.
GET_NOUN_NODE;

! Get an <address> - the is a status warning
! if more is on the line.
STATUS = DBG$NPARSE ADDRESS(.INPUT_DESC,
  NOUN_NODE[DBG$L_NOUN_VALUE],
  .DBG$GB_RADIX[DBG$B_RADIX_INPUT],
  TOKEN$K_TERM_COMMA, .MESSAGE_VECTOR);

! If the status indicates success, there is no
! more on the line, and we exit this loop.
IF .STATUS EQLU ST$K_SUCCESS
THEN
  EXITLOOP;

! If the status is not a warning, something is
! wrong, so return with a failure.
```

```
3290      3414      5      !
3291      3415      5      ! IF .STATUS NEQU ST$K_WARNING
3292      3416      5      THEN
3293      3417      5      RETURN .STATUS
3294      3418      5      END
3295      3419      4      WHILE (MATCH (DBG$CS_COMMA));
3296      3420      4
3297      3421      4      END;
3298      3422      4
3299      3423      4      END;
3300      3424      4
3301      3425      4
3302      3426      4      ! Parse [SHOW|CANCEL] WATCH [[ /ALL ; <address> [ , <address> ]*]].
3303      3427      4      !
3304      3428      4      [EVENT$K_SHOW_WATCH, EVENT$K_CANCEL_WATCH]:
3305      3429      4      BEGIN
3306      3430      4      IF MATCH (DBG$CS_SLASH)
3307      3431      4      THEN
3308      3432      4      BEGIN
3309      3433      4      GET_ADVERB_NODE;
3310      3434      4      NEED_MATCH (DBG$CS_ALL);
3311      3435      4      ADVERB_NODE [DBG$B_ADVERB_LITERAL] = ADVERB_ALL;
3312      3436      4      END
3313      3437      4
3314      3438      4      ELSE
3315      3439      4      BEGIN
3316      3440      4      IF NOT MATCH (DBG$CS_CR)
3317      3441      4      THEN
3318      3442      4      ! Get the <address> [ , <address> ], as long as
3319      3443      4      ! an <address> is terminated with a ','.
3320      3444      4      !
3321      3445      4      DO
3322      3446      5      BEGIN
3323      3447      5
3324      3448      5      ! Get a noun node for an <address>.
3325      3449      5      !
3326      3450      5      GET_NOUN_NODE;
3327      3451      5
3328      3452      5      ! Get an <address> - the is a status warning
3329      3453      5      ! if more is on the line.
3330      3454      5      !
3331      3455      5      STATUS = DBG$NPARSE ADDRESS(.INPUT_DESC,
3332      3456      5      NOUN_NODE[DBG$N_NOUN_VALUE],
3333      3457      5      DBG$GB_RADIX[DBG$B_RADIX_INPUT],
3334      3458      5      TOKEN$K_TERM_COMMA, .MESSAGE_VECTOR);
3335      3459      5
3336      3460      5      ! If the status indicates success, there is no
3337      3461      5      ! more on the line, and we exit this loop.
3338      3462      5      !
3339      3463      5      IF .STATUS EQLU ST$K_SUCCESS
3340      3464      5      THEN
3341      3465      5      EXITLOOP;
3342      3466      5
3343      3467      5
3344      3468      5
3345      3469      5
3346      3470      5
```



```
3347      3471      5      : If the status is not a warning, something is
3348      3472      5      : wrong, so return with a failure.
3349      3473      5      :
3350      3474      5      IF .STATUS NEQU STS$K_WARNING
3351      3475      5      THEN
3352      3476      5      RETURN .STATUS;
3353      3477      5      END
3354      3478      5      WHILE (MATCH (DBG$CS_COMMA));
3355      3479      4
3356      3480      4
3357      3481      5      END;
3358      3482      5
3359      3483      5      END;
3360      3484      5
3361      3485      5      TES;
3362      3486      5
3363      3487      5
3364      3488      5      : Check the status after parsing <address> [ , <address> ] or <mnemonic>
3365      3489      5      [ , <mnemonic> ]. If there's nothing left on the line, the status will
3366      3490      5      be success, and we can leave. Otherwise we signal a syntax error.
3367      3491      5
3368      3492      5      IF .STATUS EQLU STS$K_SUCCESS THEN RETURN STS$K_SUCCESS;
3369      3493      5      .MESSAGE VECT = DBG$N$SYNTAX_ERROR (DBG$N$NEXT_WORD (.INPUT_DESC));
3370      3494      5      RETURN STS$K_SEVERE
3371      3495      5
3372      3496      1      END;
```

```
                                .PSECT  DBG$PLIT,NOWRT,  SHR,  PIC,0
4E  4F  49  54  43  55  52  54  53  4E  49  0B  001AB P.ABT:  .ASCII  <9>\EXCEPTION\
                                .ASCII  <11>\INSTRUCTION\
                                .ASCII  <4>\CALL\
                                .ASCII  <6>\RETURN\
                                .ASCII  <6>\BRANCH\
                                .ASCII  <4>\LINE\
                                .ASCII  <3>\ALL\
                                .ASCII  <1>\,\,
                                .ASCII  <1>\,\,
                                .BYTE   1, 13
                                0D  01  001D7 P.ACC:
                                2C  01  001D3 P.ACA:
                                2F  01  001D5 P.ACB:
                                0D  01  001D7 P.ACC:
```

```
DBG$CS_EXCEPTION= P.ABT
DBG$CS_INSTRUCTION= P.ABU
DBG$CS_CALL= P.ABV
DBG$CS_RETURN= P.ABW
DBG$CS_BRANCH= P.ABX
DBG$CS_LINE= P.ABY
DBG$CS_ALL= P.ABZ
DBG$CS_COMMA= P.ACA
DBG$CS_SLASH= P.ACB
DBG$CS_CR= P.ACC
```

```
                                .PSECT  DBG$CODE,NOWRT,  SHR,  PIC,0
OFFC 00000 .ENTRY  DBG$EVENT_SHOW_CANCEL_SYNTAX, Save R2,R3,- ; 3186
```

Address	Hex	Op	Op2	Op3	Op4	Op5	Op6	Op7	Op8	Op9	Op10	Op11	Op12	Op13	Op14	Op15	Op16	Op17	Op18	Op19	Op20	Op21	Op22	Op23	Op24	Op25	Op26	Op27	Op28	Op29	Op30	Op31	Op32	Op33	Op34	Op35	Op36	Op37	Op38	Op39	Op40	Op41	Op42	Op43	Op44	Op45	Op46	Op47	Op48	Op49	Op50	Op51	Op52	Op53	Op54	Op55	Op56	Op57	Op58	Op59	Op60	Op61	Op62	Op63	Op64	Op65	Op66	Op67	Op68	Op69	Op70	Op71	Op72	Op73	Op74	Op75	Op76	Op77	Op78	Op79	Op80	Op81	Op82	Op83	Op84	Op85	Op86	Op87	Op88	Op89	Op90	Op91	Op92	Op93	Op94	Op95	Op96	Op97	Op98	Op99	Op100	Op101	Op102	Op103	Op104	Op105	Op106	Op107	Op108	Op109	Op110	Op111	Op112	Op113	Op114	Op115	Op116	Op117	Op118	Op119	Op120	Op121	Op122	Op123	Op124	Op125	Op126	Op127	Op128	Op129	Op130	Op131	Op132	Op133	Op134	Op135	Op136	Op137	Op138	Op139	Op140	Op141	Op142	Op143	Op144	Op145	Op146	Op147	Op148	Op149	Op150	Op151	Op152	Op153	Op154	Op155	Op156	Op157	Op158	Op159	Op160	Op161	Op162	Op163	Op164	Op165	Op166	Op167	Op168	Op169	Op170	Op171	Op172	Op173	Op174	Op175	Op176	Op177	Op178	Op179	Op180	Op181	Op182	Op183	Op184	Op185	Op186	Op187	Op188	Op189	Op190	Op191	Op192	Op193	Op194	Op195	Op196	Op197	Op198	Op199	Op200	Op201	Op202	Op203	Op204	Op205	Op206	Op207	Op208	Op209	Op210	Op211	Op212	Op213	Op214	Op215	Op216	Op217	Op218	Op219	Op220	Op221	Op222	Op223	Op224	Op225	Op226	Op227	Op228	Op229	Op230	Op231	Op232	Op233	Op234	Op235	Op236	Op237	Op238	Op239	Op240	Op241	Op242	Op243	Op244	Op245	Op246	Op247	Op248	Op249	Op250	Op251	Op252	Op253	Op254	Op255	Op256	Op257	Op258	Op259	Op260	Op261	Op262	Op263	Op264	Op265	Op266	Op267	Op268	Op269	Op270	Op271	Op272	Op273	Op274	Op275	Op276	Op277	Op278	Op279	Op280	Op281	Op282	Op283	Op284	Op285	Op286	Op287	Op288	Op289	Op290	Op291	Op292	Op293	Op294	Op295	Op296	Op297	Op298	Op299	Op300	Op301	Op302	Op303	Op304	Op305	Op306	Op307	Op308	Op309	Op310	Op311	Op312	Op313	Op314	Op315	Op316	Op317	Op318	Op319	Op320	Op321	Op322	Op323	Op324	Op325	Op326	Op327	Op328	Op329	Op330	Op331	Op332	Op333	Op334	Op335	Op336	Op337	Op338	Op339	Op340	Op341	Op342	Op343	Op344	Op345	Op346	Op347	Op348	Op349	Op350	Op351	Op352	Op353	Op354	Op355	Op356	Op357	Op358	Op359	Op360	Op361	Op362	Op363	Op364	Op365	Op366	Op367	Op368	Op369	Op370	Op371	Op372	Op373	Op374	Op375	Op376	Op377	Op378	Op379	Op380	Op381	Op382	Op383	Op384	Op385	Op386	Op387	Op388	Op389	Op390	Op391	Op392	Op393	Op394	Op395	Op396	Op397	Op398	Op399	Op400	Op401	Op402	Op403	Op404	Op405	Op406	Op407	Op408	Op409	Op410	Op411	Op412	Op413	Op414	Op415	Op416	Op417	Op418
---------	-----	----	-----	-----	-----	-----	-----	-----	-----	-----	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------

	F3	A9	9F	000AC	PUSHAB	DBG\$CS_LINE	
		54	DD	000AF	PUSHL	R4	
68		03	FB	000B1	CALLS	#3, DBG\$NMATCH	
01		50	D1	000B4	CMPL	R0, #1	
		05	12	000B7	BNEQ	7\$	
62		14	90	000B9	MOVB	#20, (ADVERB_NODE)	3348
		12	11	000BC	BRB	8\$	
		01	DD	000BE	PUSHL	#1	3354
	D4	A9	9F	000C0	PUSHAB	DBG\$CS_INSTRUCTION	
		54	DD	000C3	PUSHL	R4	
68		03	FB	000C5	CALLS	#3, DBG\$NMATCH	
01		50	D1	000C8	CMPL	R0, #1	
		06	12	000CB	BNEQ	9\$	
62		15	90	000CD	MOVB	#21, (ADVERB_NODE)	3355
	010B	31	000D0	BRW	22\$		
	01	DD	000D3	PUSHL	#1		3361
	F8	A9	9F	000D5	PUSHAB	DBG\$CS_ALL	
		54	DD	000D8	PUSHL	R4	
68		03	FB	000DA	CALLS	#3, DBG\$NMATCH	
01		50	D1	000DD	CMPL	R0, #1	
		03	13	000E0	BEQL	10\$	
	0087	31	000E2	BRW	15\$		
	00A3	31	000E5	BRW	17\$		
	01	DD	000E8	PUSHL	#1		3385
	0210	8F	BB	000EA	PUSHR	#^M<R4,R9>	
68		03	FB	000EE	CALLS	#3, DBG\$NMATCH	
DC		50	E8	000F1	BLBS	R0, 8\$	
		04	DD	000F4	PUSHL	#4	3388
6A		01	FB	000F6	CALLS	#1, DBG\$GET_TEMPMEM	
53		50	D0	000F9	MOVL	R0, NOUN_NODE	
	08	A3	D4	000FC	CLRL	8(NOUN_NODE)	
66		53	D0	000FF	MOVL	NOUN_NODE, (NOUN_LINK)	
56		A3	9E	00102	MOVAB	8(NOUN_NODE), NOUN_LINK	
	0C	AC	DD	00106	PUSHL	MESSAGE_VECT	3401
		01	DD	00109	PUSHL	#1	3399
7E		6B	9A	0010B	MOVZBL	DBG\$GB_RADIX, -(SP)	3400
		53	DD	0010E	PUSHL	NOUN_NODE	3399
		54	DD	00110	PUSHL	R4	
00000000G	00	05	FB	00112	CALLS	#5, DBG\$NPARSE_ADDRESS	
	57	50	D0	00119	MOVL	R0, STATUS	
	01	57	D1	0011C	CMPL	STATUS, #1	3407
		AF	13	0011F	BEQL	8\$	
		57	D5	00121	TSTL	STATUS	3415
		03	13	00123	BEQL	13\$	
	00A5	31	00125	BRW	20\$		
	01	DD	00128	PUSHL	#1		3419
	FC	A9	9F	0012A	PUSHAB	DBG\$CS_COMMA	
		54	DD	0012D	PUSHL	R4	
68		03	FB	0012F	CALLS	#3, DBG\$NMATCH	
9B		50	E9	00132	BLBC	R0, 8\$	
		BD	11	00135	BRB	12\$	
0E		50	91	00137	CMPB	R0, #14	3428
		94	12	0013A	BNEQ	8\$	
		01	DD	0013C	PUSHL	#1	3430
	FE	A9	9F	0013E	PUSHAB	DBG\$CS_SLASH	
54		04	AC	D0	00141	MOVL	INPUT_DESC, R4
		54	DD	00145	PUSHL	R4	

68		03	FB	00147	CALLS	#3, DBG\$NMATCH	
43		50	E9	0014A	BLBC	R0, 18\$	
		03	DD	0014D	PUSHL	#3	3432
6A		01	FB	0014F	CALLS	#1, DBG\$GET_TEMP MEM	
52		50	D0	00152	MOVL	R0, ADVERB_NODE	
	08	A2	D4	00155	CLRL	8(ADVERB_NODE)	
65		52	D0	00158	MOVL	ADVERB_NODE, (ADVERB_LINK)	
55		A2	9E	0015B	MOVAB	8(ADVERB_NODE), ADVERB_LINK	
		01	DD	0015F	PUSHL	#1	3434
	F8	A9	9F	00161	PUSHAB	DBG\$CS_ALL	
		54	DD	00164	PUSHL	R4	
68		03	FB	00166	CALLS	#3, DBG\$NMATCH	
1F		50	E8	00169	BLBS	R0, 17\$	
		01	DD	0016C	PUSHL	#1	
	0210	8F	BB	0016E	PUSHR	#M<R4,R9>	
68		03	FB	00172	CALLS	#3, DBG\$NMATCH	
0F		50	E9	00175	BLBC	R0, 16\$	
	000280D0	8F	DD	00178	PUSHL	#164048	
00000000G	00	01	FB	0017E	CALLS	#1, DBG\$NMAKE_ARG_VECT	
		73	11	00185	BRB	25\$	
		54	DD	00187	PUSHL	R4	
		5F	11	00189	BRB	24\$	
62		17	90	0018B	MOVB	#23, (ADVERB_NODE)	3435
		4E	11	0018E	BRB	22\$	3430
		01	DD	00190	PUSHL	#1	3440
	0210	8F	BB	00192	PUSHR	#M<R4,R9>	
68		03	FB	00196	CALLS	#3, DBG\$NMATCH	
42		50	E8	00199	BLBS	R0, 22\$	
		04	DD	0019C	PUSHL	#4	3446
6A		01	FB	0019E	CALLS	#1, DBG\$GET_TEMP MEM	
53		50	D0	001A1	MOVL	R0, NOUN_NODE	
	08	A3	D4	001A4	CLRL	8(NOUN_NODE)	
66		53	D0	001A7	MOVL	NOUN_NODE, (NOUN_LINK)	
56		A3	9E	001AA	MOVAB	8(NOUN_NODE), NOUN_LINK	
	0C	AC	DD	001AE	PUSHL	MESSAGE_VECT	3460
		01	DD	001B1	PUSHL	#1	3458
7E		6B	9A	001B3	MOVZBL	DBG\$GB_RADIX, -(SP)	3459
		53	DD	001B6	PUSHL	NOUN_NODE	3458
		54	DD	001B8	PUSHL	R4	
00000000G	00	05	FB	001BA	CALLS	#5, DBG\$NPARSE_ADDRESS	
57		50	D0	001C1	MOVL	R0, STATUS	
01		57	D1	001C4	CMPL	STATUS, #1	3466
		15	13	001C7	BEQL	22\$	
		57	D5	001C9	TSTL	STATUS	3474
		04	13	001CB	BEQL	21\$	
50		57	D0	001CD	MOVL	STATUS, R0	3476
			04	001D0	RET		
		01	DD	001D1	PUSHL	#1	3479
	FC	A9	9F	001D3	PUSHAB	DBG\$CS_COMMA	
		54	DD	001D6	PUSHL	R4	
68		03	FB	001D8	CALLS	#3, DBG\$NMATCH	
BE		50	E8	001DB	BLBS	R0, 19\$	
01		57	D1	001DE	CMPL	STATUS, #1	3492
		04	12	001E1	BNEQ	23\$	
50		01	D0	001E3	MOVL	#1, R0	
			04	001E6	RET		
	04	AC	DD	001E7	PUSHL	INPUT_DESC	3493


```

000000000G  00
000000000G  00
              0C  BC
                  50

```

```

01  FB 001EA 24$:  CALLS  #1, DBG$NNEXT_WORD
50  DD 001F1      PUSHL  R0
01  FB 001F3      CALLS  #1, DBG$NSYNTAX_ERROR
50  DD 001FA 25$:  MOVL   R0, @MESSAGE_VECT
04  DD 001FE      MOVL   #4, R0
    DD 00201      RET

```

3494
3496

```
; Routine Size: 514 bytes,   Routine Base: DBG$CODE + 13E6
```

```
3374 3497 1 GLOBAL ROUTINE DBG$EVENT_SHOW_CANCEL_SEMANTICS(VERB_NODE, MESSAGE_VECT) =
3375 3498 1
3376 3499 1 FUNCTION
3377 3500 1     This routine builds the event data structures based on the
3378 3501 1     parse tree passed via the VERB_NODE address. This routine
3379 3502 1     is called from DBGNSET.
3380 3503 1
3381 3504 1 INPUTS
3382 3505 1     VERB_NODE:      A pointer to a DEBUG verb node, which defines,
3383 3506 1                     along with the verb's descendants, the desired
3384 3507 1                     command.
3385 3508 1     MESSAGE_VECT:   The address of a longword to contain the address
3386 3509 1                     of a message argument vector.
3387 3510 1
3388 3511 1 OUTPUTS
3389 3512 1     The event data structure(s) are updated to include the new event,
3390 3513 1     and may include:
3391 3514 1
3392 3515 1         EVENT$CMD_QUEUE,
3393 3516 1         EVENT$WHEN_QUEUE,
3394 3517 1         EVENT$DO_LIST_QUEUE
3395 3518 1
3396 3519 1
3397 3520 2 BEGIN
3398 3521 2
3399 3522 2 MAP
3400 3523 2     VERB_NODE: REF DBG$VERB_NODE;      ! Verb node pointer
3401 3524 2
3402 3525 2 MACRO
3403 3526 2     SHOW_OR_DELETE_EVENT_ENTRY (EVENT_ENTRY) =
3404 3527 2     BEGIN
3405 3528 2         IF .VERB_NODE [DBG$B_VERB_LITERAL] EQLU DBG$K_SHOW_VERB
3406 3529 2         THEN
3407 3530 2             SHOW_EVENT_ENTRY (.EVENT_ENTRY)
3408 3531 2
3409 3532 2         ELSE
3410 3533 2             EVENT_ENTRY [EVENT$V_DELETED] = TRUE;
3411 3534 2
3412 3535 2         SHOWED_CANCELED_ONE = TRUE;
3413 3536 2         END
3414 3537 2     %;
3415 3538 2
3416 3539 2 LOCAL
3417 3540 2     DUMMY,
3418 3541 2     EVENT_QUEUE: QUEUE HEAD,           ! Working queue head
3419 3542 2     QUEUE_ENTRY: REF EVENT$EVENT_DESCRIPTOR, ! Event queue entry pointer
3420 3543 2     NEXTQ_ENTRY: REF EVENT$EVENT_DESCRIPTOR, ! Event queue entry pointer
3421 3544 2     EVENT_ENTRY: REF EVENT$EVENT_DESCRIPTOR, ! Event entry pointer
3422 3545 2     DO_ENTRY: REF EVENT$DO_LIST_DESCRIPTOR, ! DO List entry pointer
3423 3546 2     WHEN_ENTRY: REF EVENT$WHEN_DESCRIPTOR, ! WHEN entry pointer
3424 3547 2     ADVERB_NODE: REF DBG$ADVERB_NODE,      ! Adverb node link
3425 3548 2     NOUN_NODE: REF DBG$NOUN_NODE,          ! Noun node link
3426 3549 2     PRIMARY: REF DBG$PRIMARY,              ! Event primary descriptor
3427 3550 2     VALUE: REF DBG$VALDESC,                ! Event value descriptor
3428 3551 2     ADDRESS: VECTOR [2],                  ! Event address
3429 3552 2     ADR_TYP,                             ! Event address type
3430 3553 2     SHOW_CANCEL_ALL,
```



```

3431      3554      2      SHOWED_CANCELED_ANY,
3432      3555      2      SHOWED_CANCELED_ONE,
3433      3556      2      COUNTER;
3434      3557      2
3435      3558      2
3436      3559      2
3437      3560      2      ! Initialize the working queue to be empty.
3438      3561      2
3439      3562      2      EVENT_QUEUE [L_QUEUE_FLINK] = EVENT_QUEUE;
3440      3563      2      EVENT_QUEUE [L_QUEUE_BLINK] = EVENT_QUEUE;
3441      3564      2
3442      3565      2
3443      3566      2      ! Get an event descriptor from permanent memory, and
3444      3567      2      ! link it into the working queue.
3445      3568      2
3446      3569      2      EVENT_ENTRY = DBG$GET_MEMORY (EVENT$K_EVENT_DESCRIPTOR_SIZE);
3447      3570      2      INSQUE (.EVENT_ENTRY, EVENT_QUEUE);
3448      3571      2
3449      3572      2
3450      3573      2      ! Initialize the command kind to access (default).
3451      3574      2
3452      3575      2      EVENT_ENTRY [EVENT$B_CMD_KIND] = EVENT$K_KIND_ACC;
3453      3576      2
3454      3577      2
3455      3578      2      ! Set the event's command type, found in the verb.
3456      3579      2
3457      3580      2      SELECTONE .VERB_NODE [DBG$B_VERB_COMPOSITE] OF
3458      3581      2      SET
3459      3582      2
3460      3583      2      [EVENT$K_SHOW_BREAK, EVENT$K_CANCEL_BREAK]:
3461      3584      2          EVENT_ENTRY [EVENT$B_CMD_TYPE] = EVENT$K_TYPE_BREAK;
3462      3585      2
3463      3586      2      [EVENT$K_CANCEL_BREAK_EXC]:
3464      3587      2          BEGIN
3465      3588      2              EVENT_ENTRY [EVENT$B_CMD_TYPE] = EVENT$K_TYPE_BREAK;
3466      3589      2              EVENT_ENTRY [EVENT$B_CMD_KIND] = EVENT$K_KIND_EXC;
3467      3590      2          END;
3468      3591      2
3469      3592      2      [EVENT$K_SHOW_TRACE, EVENT$K_CANCEL_TRACE]:
3470      3593      2          EVENT_ENTRY [EVENT$B_CMD_TYPE] = EVENT$K_TYPE_TRACE;
3471      3594      2
3472      3595      2      [EVENT$K_SHOW_WATCH, EVENT$K_CANCEL_WATCH]:
3473      3596      2          EVENT_ENTRY [EVENT$B_CMD_TYPE] = EVENT$K_TYPE_WATCH;
3474      3597      2
3475      3598      2      [OTHERWISE]:
3476      3599      2          $DBG_ERROR('DBGEVENT\EVENT_SHOW_CANCEL_SEMANTICS 10');
3477      3600      2
3478      3601      2      TES;
3479      3602      2
3480      3603      2
3481      3604      2      ! Initialize Has Value Descriptor flag.
3482      3605      2
3483      3606      2      EVENT_ENTRY [EVENT$V_HAS_VAL_DSCR] = FALSE;
3484      3607      2
3485      3608      2
3486      3609      2      ! Initialize the Primary pointer to NULL.
3487      3610      2

```

```

! Did we show/cancel any yet ?
! Did we show/cancel one yet ?
! Counter variable

```

```
.. 3488      3611 2      EVENT_ENTRY [EVENT$$_PRIMARY] = 0;
.. 3489      3612 2
.. 3490      3613 2
.. 3491      3614 2      ! Point to the verb's noun list.
.. 3492      3615 2
.. 3493      3616 2      NOUN_NODE = .VERB_NODE [DBG$_VERB_OBJECT_PTR];
.. 3494      3617 2
.. 3495      3618 2
.. 3496      3619 2      ! Point to the verb's adverb list.
.. 3497      3620 2
.. 3498      3621 2      ADVERB_NODE = .VERB_NODE [DBG$_VERB_ADVERB_PTR];
.. 3499      3622 2
.. 3500      3623 2
.. 3501      3624 2      ! Initialize the All flag.
.. 3502      3625 2
.. 3503      3626 2      SHOW_CANCEL_ALL = FALSE;
.. 3504      3627 2
.. 3505      3628 2
.. 3506      3629 2      ! If an adverb exists, process it.
.. 3507      3630 2
.. 3508      3631 2      IF .ADVERB_NODE NEQA 0
.. 3509      3632 2      THEN
.. 3510      3633 2          BEGIN
.. 3511      3634 2
.. 3512      3635 2
.. 3513      3636 2          ! Process adverbs based on the literal set by the
.. 3514      3637 2          ! syntactic processing.
.. 3515      3638 2
.. 3516      3639 2          SELECTONE .ADVERB_NODE [DBG$_ADVERB_LITERAL] OF
.. 3517      3640 2              SET
.. 3518      3641 2
.. 3519      3642 2
.. 3520      3643 2              ! Process the /EXCEPTION adverb.
.. 3521      3644 2              !
.. 3522      3645 2              [ADVERB_EXCEPTION]:
.. 3523      3646 2                  EVENT_ENTRY [EVENT$_CMD_KIND] = EVENT$_KIND_EXC;
.. 3524      3647 2
.. 3525      3648 2
.. 3526      3649 2              ! Process the /CALL, /RETURN, /BRANCH, /LINE, /OPCODE
.. 3527      3650 2              ! and /INSTRUCTION instruction-kind adverbs.
.. 3528      3651 2              !
.. 3529      3652 2              [ADVERB_CALL, ADVERB_RETURN, ADVERB_BRANCH,
.. 3530      3653 2              ADVERB_LINE, ADVERB_OPCODE, ADVERB_INSTRUCTION]:
.. 3531      3654 2                  BEGIN
.. 3532      3655 2
.. 3533      3656 2
.. 3534      3657 2                  ! Define the event kind to be instruction.
.. 3535      3658 2                  !
.. 3536      3659 2                  EVENT_ENTRY [EVENT$_CMD_KIND] = EVENT$_KIND_INS;
.. 3537      3660 2
.. 3538      3661 2
.. 3539      3662 2                  ! Define the subkind according to the adverb.
.. 3540      3663 2                  !
.. 3541      3664 2                  EVENT_ENTRY [EVENT$_SUB_KIND] =
.. 3542      3665 2                      (SELECTONE .ADVERB_NODE [DBG$_ADVERB_LITERAL] OF
.. 3543      3666 2                          SET
.. 3544      3667 2                          [ADVERB_CALL]:
```



```

3545      3668      5      EVENT$K INS CALL;
3546      3669      [ADVERB BRANCH]:
3547      3670      EVENT$K INS_BRAN;
3548      3671      [ADVERB LINE]:
3549      3672      EVENT$K INS_LINE;
3550      3673      [ADVERB OP$CODE]:
3551      3674      EVENT$K INS_USER;
3552      3675      [ADVERB INSTRUCTION]:
3553      3676      EVENT$K INS_EVR;
3554      3677      [OTHERWISE]:
3555      3678      $DBG_ERROR('DBGEVENT\EVENT_SHOW_CANCEL_SEMANTICS 20');
3556      3679      TES
3557      3680      )
3558      3681      END;
3559      3682
3560      3683      ! Process the /ALL adverb.
3561      3684      !
3562      3685      [ADVERB ALL]:
3563      3686      SHOW_CANCEL_ALL = TRUE;
3564      3687
3565      3688      ! An invalid adverb literal - yell !
3566      3689      !
3567      3690      [OTHERWISE]:
3568      3691      $DBG_ERROR('DBGEVENT\EVENT_SHOW_CANCEL_SEMANTICS 30');
3569      3692
3570      3693      TES;
3571      3694
3572      3695      END;
3573      3696
3574      3697
3575      3698
3576      3699
3577      3700      ! If there are no noun nodes, set the All flag if this is a
3578      3701      ! SHOW command. Otherwise, make sure the command kind is
3579      3702      ! (still) access, and process the address(es).
3580      3703      !
3581      3704      IF .NOUN_NODE EQLA 0
3582      3705      THEN
3583      3706      BEGIN
3584      3707      IF .VERB_NODE [DBG$B_VERB_LITERAL] EQLU DBG$K_SHOW_VERB
3585      3708      THEN
3586      3709      SHOW_CANCEL_ALL = TRUE;
3587      3710
3588      3711      END
3589      3712
3590      3713      ELSE
3591      3714      BEGIN
3592      3715
3593      3716
3594      3717      ! Build duplicate event entries, one for each
3595      3718      ! address found in the noun node list.
3596      3719      !
3597      3720      WHILE TRUE DO
3598      3721      BEGIN
3599      3722
3600      3723
3601      3724      ! Using the address's primary descriptor, get the address.
```

```

: 3602
: 3603
: 3604
: 3605
: 3606
: 3607
: 3608
: 3609
: 3610
: 3611
: 3612
: 3613
: 3614
: 3615
: 3616
: 3617
: 3618
: 3619
: 3620
: 3621
: 3622
: 3623
: 3624
: 3625
: 3626
: 3627
: 3628
: 3629
: 3630
: 3631
: 3632
: 3633
: 3634
: 3635
: 3636
: 3637
: 3638
: 3639
: 3640
: 3641
: 3642
: 3643
: 3644
: 3645
: 3646
: 3647
: 3648
: 3649
: 3650
: 3651
: 3652
: 3653
: 3654
: 3655
: 3656
: 3657
: 3658

```

```

3725 4
3726 4
3727 4
3728 4
3729 4
3730 4
3731 5
3732 5
3733 5
3734 6
3735 6
3736 6
3737 6
3738 6
3739 6
3740 5
3741 5
3742 4
3743 4
3744 4
3745 5
3746 5
3747 5
3748 5
3749 4
3750 4
3751 4
3752 4
3753 4
3754 4
3755 4
3756 4
3757 4
3758 5
3759 5
3760 5
3761 5
3762 5
3763 4
3764 4
3765 4
3766 4
3767 4
3768 4
3769 4
3770 4
3771 4
3772 4
3773 4
3774 4
3775 4
3776 4
3777 4
3778 4
3779 4
3780 4
3781 4

```

```

!
PRIMARY = .NOUN_NODE [DBG$L NOUN_VALUE];
EVENT_ENTRY [EVENT$V BREAK_ADDRESS] = FALSE;
IF .EVENT_ENTRY [EVENT$B_CMD_TYPE] EQL EVENT$K_TYPE_BREAK OR
.EVENT_ENTRY [EVENT$B_CMD_TYPE] EQL EVENT$K_TYPE_TRACE
THEN
  BEGIN
    IF .PRIMARY[DBG$B_DHDR_TYPE] EQL DBG$K_PRIMARY_DESC
    THEN
      BEGIN
        IF .PRIMARY[DBG$B_DHDR_KIND] EQL RST$K_ROUTINE OR
.PRIMARY[DBG$B_DHDR_KIND] EQL RST$K_ENTRY
        THEN
          EVENT_ENTRY [EVENT$V_BREAK_ADDRESS] = TRUE;
        END;
      END;
    END;
  IF NOT DBG$NGET_ADDRESS(.PRIMARY, ADDRESS[0], ADR_TYP,
    (IF .EVENT_ENTRY [EVENT$V_BREAK_ADDRESS]
    THEN
      TRUE
    ELSE
      FALSE),
    .MESSAGE_VECT)
  THEN
    RETURN ST$K_SEVERE;

  EVENT_ENTRY [EVENT$L_ADDRESS] = .ADDRESS[0];
  IF NOT .EVENT_ENTRY [EVENT$V_BREAK_ADDRESS]
  THEN
    BEGIN
      IF DBG$IS_IT_ENTRY (.ADDRESS [0])
      THEN
        EVENT_ENTRY [EVENT$L_ADDRESS] = .ADDRESS[0] + 2;
      END;

    ! Save the event's primary.
    !
    EVENT_ENTRY [EVENT$L_PRIMARY] = .NOUN_NODE [DBG$L NOUN_VALUE];
    DBG$NCOPY_DESC(.EVENT_ENTRY [EVENT$L_PRIMARY], PRIMARY, DUMMY, FALSE);

    ! Now, get and save the VMS descriptor.
    !
    DBG$PRIM TO_VAL(.PRIMARY, DBG$K_V_VALUE_DESC, VALUE);
    CH$MOVE(T2, VALUE[DBG$A_VALUE_VMSDESC],
      EVENT_ENTRY[EVENT$A_VMSDESC]);

    ! Point to the next noun node.
    !
    NOUN_NODE = .NOUN_NODE[DBG$L_NOUN_LINK];

```



```

: 3659      3782  4
: 3660      3783  4
: 3661      3784  4
: 3662      3785  4
: 3663      3786  4
: 3664      3787  4
: 3665      3788  4
: 3666      3789  4
: 3667      3790  5
: 3668      3791  5
: 3669      3792  5
: 3670      3793  5
: 3671      3794  5
: 3672      3795  5
: 3673      3796  5
: 3674      3797  5
: 3675      3798  5
: 3676      3799  5
: 3677      3800  5
: 3678      3801  5
: 3679      3802  5
: 3680      3803  5
: 3681      3804  5
: 3682      3805  5
: 3683      3806  4
: 3684      3807  4
: 3685      3808  4
: 3686      3809  4
: 3687      3810  4
: 3688      3811  2
: 3689      3812  2
: 3690      3813  2
: 3691      3814  2
: 3692      3815  2
: 3693      3816  2
: 3694      3817  2
: 3695      3818  2
: 3696      3819  2
: 3697      3820  2
: 3698      3821  2
: 3699      3822  2
: 3700      3823  2
: 3701      3824  2
: 3702      3825  2
: 3703      3826  2
: 3704      3827  2
: 3705      3828  2
: 3706      3829  2
: 3707      3830  2
: 3708      3831  2
: 3709      3832  2
: 3710      3833  2
: 3711      3834  2
: 3712      3835  2
: 3713      3836  2
: 3714      3837  2
: 3715      3838  2

! If there's more addresses, get an new event descriptor
! from permanent memory, copy the current one to it, and
! link it to the working queue; otherwise, exit this loop.
IF .NOUN_NODE NEQA 0
THEN
  BEGIN

    ! Get another event descriptor, and copy the old
    ! one into it.
    EVENT_ENTRY = DBG$GET_MEMORY(EVENT$K_EVENT_DESCRIPTOR_SIZE);
    CH$MOVE(EVENT$K_EVENT_DESCRIPTOR_SIZE * 4,
            .EVENT_QUEUE [L_QUEUE_FLINK], .EVENT_ENTRY);

    ! Link the new one into the working queue.
    INSQUE (.EVENT_ENTRY, EVENT_QUEUE);
  END
ELSE
  EXITLOOP;
END
END;

! Now that we've built a working list of event entries, compare them to
! the Command Queue, showing or deleting existing entries that match new
! ones.
! For each new event entry, check the existing entries in the
! Command queue for any entries that match (command types, and
! addresses if the command type is an access type).
! So, while there are (still) entries on the working queue....
SHOWED_CANCELED_ANY = FALSE;
WHILE .EVENT_QUEUE [L_QUEUE_FLINK] NEQA EVENT_QUEUE DO
  BEGIN

    ! Point EVENT_ENTRY to the first new event entry on the
    ! working queue.
    EVENT_ENTRY = .EVENT_QUEUE [L_QUEUE_FLINK];

    ! Point to the first and next existing entries in the Command Queue.
    QUEUE_ENTRY = .EVENT$CMD_QUEUE [L_QUEUE_FLINK];
```

```
3716 3839 3
3717 3840 3
3718 3841 3
3719 3842 3
3720 3843 3
3721 3844 4
3722 3845 4
3723 3846 4
3724 3847 4
3725 3848 4
3726 3849 4
3727 3850 4
3728 3851 5
3729 3852 5
3730 3853 5
3731 3854 5
3732 3855 5
3733 3856 5
3734 3857 5
3735 3858 5
3736 3859 6
3737 3860 6
3738 3861 6
3739 3862 6
3740 3863 6
3741 3864 6
3742 3865 6
3743 3866 6
3744 3867 6
3745 3868 6
3746 3869 6
3747 3870 6
3748 3871 6
3749 3872 6
3750 3873 6
3751 3874 6
3752 3875 7
3753 3876 7
3754 3877 6
3755 3878 7
3756 3879 7
3757 3880 7
3758 3881 7
3759 3882 7
3760 3883 7
3761 3884 7
3762 3885 7
3763 3886 7
3764 3887 8
3765 3888 8
3766 3889 8
3767 3890 8
3768 3891 8
3769 3892 8
3770 3893 8
3771 3894 8
3772 3895 8
```

```
! For each existing entry...
SHOWED_CANCELED_ONE = FALSE;
WHILE .QUEUE_ENTRY NEQA EVENT$CMD_QUEUE DO
  BEGIN

    ! Ignore 'deleted' entries that haven't been removed.
    IF NOT .QUEUE_ENTRY [EVENT$V_DELETED]
    THEN
      BEGIN

        ! Compare the command type with the existing one.
        IF .EVENT_ENTRY [EVENT$B_CMD_TYPE] EQLU
        .QUEUE_ENTRY [EVENT$B_CMD_TYPE]
        THEN
          BEGIN

            ! We found an existing entry of the right type, so
            ! there is/are BREAK(s), TRACE(s), or WATCH(es) to
            ! be checked.
            SHOWED_CANCELED_ANY = TRUE;

            ! If the /ALL qualifier was used, the show or delete
            ! this entry. Otherwise, we've got more checking to
            ! do....
            IF .SHOW_CANCEL_ALL
            THEN
              SHOW_OR_DELETE_EVENT_ENTRY (QUEUE_ENTRY)
            ELSE
              BEGIN

                ! Compare the command kind of this existing entry
                ! and the new one.
                IF .EVENT_ENTRY [EVENT$B_CMD_KIND] EQLU
                .QUEUE_ENTRY [EVENT$B_CMD_KIND]
                THEN
                  BEGIN

                    ! The command kinds match. Select on the kind.
                    SELECTONE .EVENT_ENTRY [EVENT$B_CMD_KIND] OF
                    SET
```


3773	3896	8
3774	3897	8
3775	3898	8
3776	3899	8
3777	3900	8
3778	3901	8
3779	3902	9
3780	3903	9
3781	3904	9
3782	3905	9
3783	3906	9
3784	3907	9
3785	3908	9
3786	3909	9
3787	3910	9
3788	3911	9
3789	3912	9
3790	3913	11
3791	3914	11
3792	3915	11
3793	3916	11
3794	3917	10
3795	3918	10
3796	3919	10
3797	3920	10
3798	3921	10
3799	3922	10
3800	3923	10
3801	3924	9
3802	3925	9
3803	3926	8
3804	3927	8
3805	3928	8
3806	3929	8
3807	3930	8
3808	3931	8
3809	3932	8
3810	3933	8
3811	3934	8
3812	3935	8
3813	3936	8
3814	3937	8
3815	3938	9
3816	3939	9
3817	3940	9
3818	3941	9
3819	3942	9
3820	3943	8
3821	3944	8
3822	3945	9
3823	3946	10
3824	3947	9
3825	3948	10
3826	3949	10
3827	3950	9
3828	3951	9
3829	3952	8

! The command kind is access (/READ, /WRITE,
/MODIFY, /EXECUTE, or /RETURN). We now have
to compare the addresses of the entries to
determine if this existing one must be deleted.

[EVENTSK_KIND_ACC]:

BEGIN

LOCAL CODE_1, CODE_2;

! If either of the entries is /EXECUTE or
/RETURN, we'll compare the byte addresses,
otherwise compare the two VMS descriptors,
and delete the existing entry if they're
the same.

CODE_1 = .EVENT_ENTRY [EVENTSB_SUB_KIND];

CODE_2 = .QUEUE_ENTRY [EVENTSB_SUB_KIND];

IF (IF (.CODE_1 EQLU EVENTSK_ACC_EXEC)

OR (.CODE_1 EQLU EVENTSK_ACC_RTRN)

OR (.CODE_2 EQLU EVENTSK_ACC_EXEC)

OR (.CODE_2 EQLU EVENTSK_ACC_RTRN)

THEN

.EVENT_ENTRY [EVENTSL_ADDRESS] EQLA

.QUEUE_ENTRY [EVENTSL_ADDRESS]

ELSE

COMPARE_VMSDESC (EVENT_ENTRY [EVENTSA_VMSDESC],
QUEUE_ENTRY [EVENTSA_VMSDESC])

)

THEN

SHOW_OR_DELETE_EVENT_ENTRY (QUEUE_ENTRY);

END;

! The command kind is instruction. Compare the two
entries' subkind, and delete the existing entry if
they're the same.

[EVENTSK_KIND_INS]:

CASE .EVENT_ENTRY [EVENTSB_SUB_KIND]

FROM EVENTSK_INS_CALL TO EVENTSK_INS_USER OF

SET

[EVENTSK_INS_CALL TO EVENTSK_INS_LINE]:

BEGIN

IF .EVENT_ENTRY [EVENTSB_SUB_KIND] EQLU

.QUEUE_ENTRY [EVENTSB_SUB_KIND]

THEN

SHOW_OR_DELETE_EVENT_ENTRY (QUEUE_ENTRY);

END;

[EVENTSK_INS_EVERY TO EVENTSK_INS_USER]:

BEGIN

IF (.QUEUE_ENTRY [EVENTSB_SUB_KIND] EQLU

EVENTSK_INS_EVERY) OR

(.QUEUE_ENTRY [EVENTSB_SUB_KIND] EQLU

EVENTSK_INS_USER)

THEN

SHOW_OR_DELETE_EVENT_ENTRY (QUEUE_ENTRY);

END;

```
3830
3831
3832
3833
3834
3835
3836
3837
3838
3839
3840
3841
3842
3843
3844
3845
3846
3847
3848
3849
3850
3851
3852
3853
3854
3855
3856
3857
3858
3859
3860
3861
3862
3863
3864
3865
3866
3867
3868
3869
3870
3871
3872
3873
3874
3875
3876
3877
3878
3879
3880
3881
3882
3883
3884
3885
3886
```

```
3953
3954
3955
3956
3957
3958
3959
3960
3961
3962
3963
3964
3965
3966
3967
3968
3969
3970
3971
3972
3973
3974
3975
3976
3977
3978
3979
3980
3981
3982
3983
3984
3985
3986
3987
3988
3989
3990
3991
3992
3993
3994
3995
3996
3997
3998
3999
4000
4001
4002
4003
4004
4005
4006
4007
4008
4009
```

```
TES;

! The access type is exception, so delete the existing
! entry.
[EVENT$K_KIND_EXC]:
    SHOW_OR_DELETE_EVENT_ENTRY (QUEUE_ENTRY);

! The command kind is invalid - yell !
[OTHERWISE]:
    $DBG_ERROR('DBGEVENT\EVENT_SHOW_CANCEL_SEMANTICS 40');

TES;

END;

END;

END;

END;

! Point to the next Command Queue entries.
!
QUEUE_ENTRY = .QUEUE_ENTRY [EVENT$L_CMD_FLINK];
END;

! If this entry wasn't shown or cancelled, announce the fact, but we
! did find one of this type, say we couldn't find this one.
IF NOT .SHOWED_CANCELED_ONE AND .SHOWED_CANCELED_ANY
THEN
    SIGNAL      (SELECT ONE .VERB_NODE [DBG$B_VERB_COMPOSITE] OF
                SET
                [EVENT$K_SHOW_BREAK, EVENT$K_CANCEL_BREAK,
                EVENT$K_CANCEL_BREAK_EXC]:
                DBG$NOSUCHBPT;
                [EVENT$K_SHOW_TRACE, EVENT$K_CANCEL_TRACE]:
                DBG$NOSUCHTPT;
                [EVENT$K_SHOW_WATCH, EVENT$K_CANCEL_WATCH]:
                DBG$NOSUCHWPT;
                TES
                );

! Remove this new entry from the working queue.
!
REMQUE (.EVENT_ENTRY, EVENT_ENTRY);
DBG$REL_MEMORY(.EVENT_ENTRY);
END;
```


DBGEVENT
V04-000

N 13
16-Sep-1984 00:59:10 VAX-11 Bliss-32 V4.0-742
14-Sep-1984 12:16:53 [DEBUG.SRC]DBGEVENT.B32;1

Page 109
(11)

```
: 3887      4010      2      ! If we couldn't find an entry of the right type, announce the fact.
: 3888      4011      2      !
: 3889      4012      2      IF NOT .SHOWED_CANCELED_ANY
: 3890      4013      2      THEN
: 3891      4014      2      SIGNAL (SELECTONE .VERB_NODE [DBG$B_VERB_COMPOSITE] OF
: 3892      4015      2      SET
: 3893      4016      2      [EVENT$K_SHOW_BREAK, EVENT$K_CANCEL_BREAK,
: 3894      4017      2      EVENT$K_CANCEL_BREAK_EXC]:
: 3895      4018      2      DBGS_NOBREAKS;
: 3896      4019      2      [EVENT$K_SHOW_TRACE, EVENT$K_CANCEL_TRACE]:
: 3897      4020      2      DBGS_NOTRACES;
: 3898      4021      2      [EVENT$K_SHOW_WATCH, EVENT$K_CANCEL_WATCH]:
: 3899      4022      2      DBGS_NOWATCHES;
: 3900      4023      2      TES
: 3901      4024      2      );
: 3902      4025      2
: 3903      4026      2
: 3904      4027      2      ! We're done, bye.
: 3905      4028      2      !
: 3906      4029      2      RETURN ST$K_SUCCESS;
: 3907      4030      1      END;
```

```
.PSECT DBGSPLIT,NOWRT, SHR, PIC,0

54 4E 45 56 45 5C 54 4E 45 56 45 47 42 44 27 001D9 P.ACD: .ASCII \DBGEVENT\<92>\EVENT_SHOW_CANCEL_SEMANT\
45 53 5F 4C 45 43 4E 41 43 5F 57 4F 48 53 5F 001E8
: 30 31 20 53 43 49 001F7
54 4E 45 56 45 5C 54 4E 45 56 45 47 42 44 27 00201 P.ACE: .ASCII \ICS 10\
45 53 5F 4C 45 43 4E 41 43 5F 57 4F 48 53 5F 00210
: 54 4E 45 56 45 5C 54 4E 45 56 45 47 42 44 27 00223
45 53 5F 4C 45 43 4E 41 43 5F 57 4F 48 53 5F 00229 P.ACF: .ASCII \ICS 20\
: 54 4E 45 56 45 5C 54 4E 45 56 45 47 42 44 27 00238
45 53 5F 4C 45 43 4E 41 43 5F 57 4F 48 53 5F 00247
: 30 33 20 53 43 49 0024B
54 4E 45 56 45 5C 54 4E 45 56 45 47 42 44 27 00251 P.ACG: .ASCII \ICS 30\
45 53 5F 4C 45 43 4E 41 43 5F 57 4F 48 53 5F 00260
: 54 4E 45 56 45 5C 54 4E 45 56 45 47 42 44 27 0026F
45 53 5F 4C 45 43 4E 41 43 5F 57 4F 48 53 5F 00273
: 30 34 20 53 43 49 00273
: .ASCII \ICS 40\
:
```

```
.PSECT DBG$CODE,NOWRT, SHR, PIC,0

OFFC 00000

5B 00000000' EF 9E 00002
5A 00000000' EF 9E 00009
59 00000000G 00 9E 00010
SE 28 C2 00017
18 AE 18 AE 9E 0001A
1C AE 18 AE 9E 0001F
00000000G 00 10 DD 00024
01 FB 00026

.ENTRY DBG$EVENT_SHOW_CANCEL_SEMANTICS, Save R2,- 3497
R3,R4,R5,R6,R7,R8,R9,R10,R11
MOVAB EVENT$CMD_QUEUE, R11
MOVAB P.ACD, R10
MOVAB LIB$SIGNAL, R9
SUBL2 #40, SP
MOVAB EVENT_QUEUE, EVENT_QUEUE
MOVAB EVENT_QUEUE, EVENT_QUEUE+4
PUSHL #16
CALLS #1, DBG$GET_MEMORY 3562
3563
3569
```


18	56		50	D0	0002D	MOVL	R0, EVENT_ENTRY	3570
	AE		66	0E	00030	INSQUE	(EVENT_ENTRY), EVENT_QUEUE	3575
	53	14	A6	9E	00034	MOVAB	20(EVENT_ENTRY), R3	
		01	A3	94	00038	CLRB	1(R3)	
	52	04	AC	D0	0003B	MOVL	VERB_NODE, R2	3580
	50	01	A2	9A	0003F	MOVZBL	1(R2), R0	
			09	15	00043	BLEQ	1\$	3583
	02		50	91	00045	CMPB	R0, #2	
			04	1A	00048	BGTRU	1\$	
			63	94	0004A	CLRB	(R3)	3584
			32	11	0004C	BRB	6\$	
	04		50	91	0004E	CMPB	R0, #4	3586
			07	12	00051	BNEQ	2\$	
	63	0200	8F	B0	00053	MOVW	#512, (R3)	3588
			26	11	00058	BRB	6\$	3580
	09		50	91	0005A	CMPB	R0, #9	3592
			05	13	0005D	BEQL	3\$	
	0B		50	91	0005F	CMPB	R0, #11	
			05	12	00062	BNEQ	4\$	
	63		01	90	00064	MOVB	#1, (R3)	3593
			17	11	00067	BRB	6\$	
	0E		50	91	00069	CMPB	R0, #14	3595
			05	12	0006C	BNEQ	5\$	
	63		02	90	0006E	MOVB	#2, (R3)	3596
			0D	11	00071	BRB	6\$	
			5A	DD	00073	PUSHL	R10	3599
			01	DD	00075	PUSHL	#1	
		00028362	8F	DD	00077	PUSHL	#164706	
	69		03	FB	0007D	CALLS	#3, LIB\$SIGNAL	
03	A3	28	08	8A	00080	BICB2	#8, 3(R3)	3606
		08	A6	D4	00084	CLRL	40(EVENT_ENTRY)	3611
	57	04	A2	D0	00087	MOVL	8(R2), NOUN_NODE	3616
	50		A2	D0	0008B	MOVL	4(R2), ADVERB_NODE	3621
			58	D4	0008F	CLRL	SHOW_CANCEL_ACL	3626
			50	D5	00091	TSTL	ADVERB_NODE	3631
			7C	13	00093	BEQL	17\$	
	0B		60	91	00095	CMPB	(ADVERB_NODE), #11	3645
			06	12	00098	BNEQ	7\$	
01	A3		02	90	0009A	MOVB	#2, 1(R3)	3646
			71	11	0009E	BRB	17\$	
	0A		60	91	000A0	CMPB	(ADVERB_NODE), #10	3652
			0A	13	000A3	BEQL	8\$	
	12		60	91	000A5	CMPB	(ADVERB_NODE), #18	
			4F	1F	000A8	BLSSU	15\$	
	16		60	91	000AA	CMPB	(ADVERB_NODE), #22	
			4A	1A	000AD	BGTRU	15\$	
01	A3		01	90	000AF	MOVB	#1, 1(R3)	3659
	12		60	91	000B3	CMPB	(ADVERB_NODE), #18	3667
			05	12	000B6	BNEQ	9\$	
	50		05	D0	000B8	MOVL	#5, R0	
			36	11	000BB	BRB	14\$	
	13		60	91	000BD	CMPB	(ADVERB_NODE), #19	3669
			05	12	000C0	BNEQ	10\$	
	50		06	D0	000C2	MOVL	#6, R0	
			2C	11	000C5	BRB	14\$	
	14		60	91	000C7	CMPB	(ADVERB_NODE), #20	3671
			05	12	000CA	BNEQ	11\$	

50		07	DO	000CC	MOVL	#7, R0	:		
		22	11	000CF	BRB	14\$:		
16		60	91	000D1	11\$: CMPB	(ADVERB_NODE), #22	:	3673	
		05	12	000D4	BNEQ	12\$:		
50		09	DO	000D6	MOVL	#9, R0	:		
		18	11	000D9	BRB	14\$:		
15		60	91	000DB	12\$: CMPB	(ADVERB_NODE), #21	:	3675	
		05	12	000DE	BNEQ	13\$:		
50		08	DO	000E0	MOVL	#8, R0	:		
		0E	11	000E3	BRB	14\$:		
	28	AA	9F	000E5	13\$: PUSHAB	P.ACE	:	3678	
		01	DD	000E8	PUSHL	#1	:		
	00028362	8F	DD	000EA	PUSHL	#164706	:		
69		03	FB	000F0	CALLS	#3, LIB\$SIGNAL	:		
02	A3	50	90	000F3	14\$: MOVB	R0, 2(R3)	:	3665	
		18	11	000F7	BRB	17\$:	3664	
17		60	91	000F9	15\$: CMPB	(ADVERB_NODE), #23	:	3686	
		05	12	000FC	BNEQ	16\$:		
58		01	DO	000FE	MOVL	#1, SHOW_CANCEL_ALL	:	3687	
		0E	11	00101	BRB	17\$:		
	50	AA	9F	00103	16\$: PUSHAB	P.ACF	:	3693	
		01	DD	00106	PUSHL	#1	:		
	00028362	8F	DD	00108	PUSHL	#164706	:		
69		03	FB	0010E	CALLS	#3, LIB\$SIGNAL	:		
		57	D5	00111	17\$: TSTL	NOUN_NODE	:	3704	
		0B	12	00113	BNEQ	19\$:		
0B		62	91	00115	CMPB	(R2), #11	:	3707	
		03	12	00118	BNEQ	18\$:		
58		01	DO	0011A	MOVL	#1, SHOW_CANCEL_ALL	:	3709	
		00C1	31	0011D	18\$: BRW	27\$:	3704	
08	AE	67	DO	00120	19\$: MOVL	(NOUN_NODE), PRIMARY	:	3726	
19	A6	20	8A	00124	BICB2	#32, 25(EVENT_ENTRY)	:	3727	
		14	A6	95	00128	TSTB	20(EVENT_ENTRY)	:	3728
		06	13	0012B	BEQL	20\$:		
01		14	A6	91	0012D	CMPB	20(EVENT_ENTRY), #1	:	3729
		1B	12	00131	BNEQ	22\$:		
50		08	AE	DO	00133	20\$: MOVL	PRIMARY, R0	:	3732
79	8F	02	A0	91	00137	CMPB	2(R0), #121	:	
		10	12	0013C	BNEQ	22\$:		
02		07	A0	91	0013E	CMPB	7(R0), #2	:	3735
		06	13	00142	BEQL	21\$:		
08		07	A0	91	00144	CMPB	7(R0), #8	:	3736
		04	12	00148	BNEQ	22\$:		
19	A6	20	88	0014A	21\$: BISB2	#32, 25(EVENT_ENTRY)	:	3738	
		08	AC	DD	0014E	22\$: PUSHL	MESSAGE_VECT	:	3750
04	19	A6	05	E1	00151	BBC	#5, 25(EVENT_ENTRY), 23\$:	3745
			01	DD	00156	PUSHL	#1	:	
			02	11	00158	BRB	24\$:	
			7E	D4	0015A	23\$: CLRL	-(SP)	:	
		08	AE	9F	0015C	24\$: PUSHAB	ADR_TYP	:	3744
		1C	AE	9F	0015F	PUSHAB	ADDRESS	:	
		18	AE	DD	00162	PUSHL	PRIMARY	:	
00000000G	00	05	FB	00165	CALLS	#5, DBG\$NGET_ADDRESS	:		
	04	50	E8	0016C	BLBS	R0, 25\$:		
	50	04	DO	0016F	MOVL	#4, R0	:	3752	
			04	00172	RET		:		
2C	A6	10	AE	DO	00173	25\$: MOVL	ADDRESS, 44(EVENT_ENTRY)	:	3755

13	19	A6	05	E0	00178	BBS	#5, 25(EVENT_ENTRY), 26\$	3756
			10	AE	DD 0017D	PUSHL	ADDRESS	3759
	00000000G	00		01	FB 00180	CALLS	#1, DBG\$IS_IT_ENTRY	
2C	A6	06		50	E9 00187	BLBC	R0, 26\$	
		10		02	C1 0018A	ADDL3	#2, ADDRESS, 44(EVENT_ENTRY)	3761
		28		67	D0 00190	MOVL	(NOUN_NODE), 40(EVENT_ENTRY)	3768
				7E	D4 00194	CLRL	-(SP)	3769
			08	AE	9F 00196	PUSHAB	DUMMY	
			10	AE	9F 00199	PUSHAB	PRIMARY	
	00000000G	00	28	A6	DD 0019C	PUSHL	40(EVENT_ENTRY)	
		7E		04	FB 0019F	CALLS	#4, DBG\$NOCOPY_DESC	3774
			0C	AE	9F 001A6	PUSHAB	VALUE	
			83	8F	9A 001A9	MOVZBL	#131, -(SP)	
	00000000G	00	10	AE	DD 001AD	PUSHL	PRIMARY	
		50		03	FB 001B0	CALLS	#3, DBG\$PRIM_TO_VAL	3775
34	A6	A0	0C	AE	D0 001B7	MOVL	VALUE, R0	3776
		57		0C	28 001BB	MOVC3	#12, 20(R0), 52(EVENT_ENTRY)	3781
			08	A7	D0 001C1	MOVL	8(NOUN_NODE), NOUN_NODE	3788
				1A	13 001C5	BEQL	27\$	3796
				10	DD 001C7	PUSHL	#16	
	00000000G	00		01	FB 001C9	CALLS	#1, DBG\$GET_MEMORY	
		56		50	D0 001D0	MOVL	R0, EVENT_ENTRY	
66		BE	0040	8F	28 001D3	MOVC3	#64, @EVENT_QUEUE, (EVENT_ENTRY)	3798
	18	AE		66	0E 001DA	INSQUE	(EVENT_ENTRY), EVENT_QUEUE	3803
	18			FF	3F 001DE	BRW	19\$	3788
				55	D4 001E1	CLRL	SHOWED_CANCELED_ANY	3824
		50	18	AE	9E 001E3	MOVAB	EVENT_QUEUE, R0	3825
		50	18	AE	D1 001E7	CMPL	EVENT_QUEUE, R0	
				03	12 001EB	BNEQ	29\$	
				011B	31 001ED	BRW	56\$	
		56	18	AE	D0 001F0	MOVL	EVENT_QUEUE, EVENT_ENTRY	3832
		52		6B	D0 001F4	MOVL	EVENT\$CMD_QUEUE, QUEUE_ENTRY	3837
				54	D4 001F7	CLRL	SHOWED_CANCELED_ONE	3842
		50		6B	9E 001F9	MOVAB	EVENT\$CMD_QUEUE, R0	3843
		50		52	D1 001FC	CMPL	QUEUE_ENTRY, R0	
				03	12 001FF	BNEQ	31\$	
				00B3	31 00201	BRW	47\$	
		53	14	A2	9E 00204	MOVAB	20(QUEUE_ENTRY), R3	3849
				63	D5 00208	TSTL	(R3)	
				03	18 0020A	BGEQ	33\$	
				00A2	31 0020C	BRW	46\$	
		63	14	A6	91 0020F	CMPB	20(EVENT_ENTRY), (R3)	3857
				F7	12 00213	BNEQ	32\$	
		55		01	D0 00215	MOVL	#1, SHOWED_CANCELED_ANY	3866
		6F		58	E8 00218	BLBS	SHOW_CANCEL_ALL, 42\$	3873
		50	15	A6	9A 0021B	MOVZBL	21(EVENT_ENTRY), R0	3884
		50	01	A3	91 0021F	CMPB	1(R3), R0	3885
				E7	12 00223	BNEQ	32\$	
				50	D5 00225	TSTL	R0	3901
				35	12 00227	BNEQ	37\$	
		50	16	A6	9A 00229	MOVZBL	22(EVENT_ENTRY), CODE_1	3911
		51	02	A3	9A 0022D	MOVZBL	2(R3), CODE_2	3912
		03		50	D1 00231	CMPL	CODE_1, #3	3913
				0F	13 00234	BEQL	34\$	
		04		50	D1 00236	CMPL	CODE_1, #4	3914
				0A	13 00239	BEQL	34\$	
		03		51	D1 0023B	CMPL	CODE_2, #3	3915

DBGEVENT
V04-000

F 14
16-Sep-1984 00:59:10
14-Sep-1984 12:16:53

VAX-11 Bliss-32 V4.0-742
[DEBUG.SRC]DBGEVENT.B32;1

Page 114
(11)

	0E	50	91	002E9	52\$:	CMPB	R0	#14	...	3997
		05	13	002EC		BEQL	53\$...	
	7E	01	CE	002EE		MNEGL	#1	-(SP)	...	
		06	11	002F1		BRB	54\$...	
		8F	DD	002F3	53\$:	PUSHL	#163883		...	
	69	01	FB	002F9	54\$:	CALLS	#1, LIB\$SIGNAL		...	3990
	56	66	OF	002FC	55\$:	REMQUE	(EVENT_ENTRY), EVENT_ENTRY		...	4005
		56	DD	002FF		PUSHL	EVENT_ENTRY		...	4006
		01	FB	00301		CALLS	#1, DBG\$REL_MEMORY		...	
		FED8	31	00308		BRW	28\$...	3825
	3F	55	E8	0030B	56\$:	BLBS	SHOWED_CANCELED_ANY, 64\$...	4012
50		08	08	EF	0030E	EXTZV	#8, #8, @VERB_NODE, R0		...	4014
		05	15	00314		BLEQ	57\$...	4016
	02	50	91	00316		CMPB	R0	#2	...	
		05	1B	00319		BLEQU	58\$...	
	04	50	91	0031B	57\$:	CMPB	R0	#4	...	
		08	12	0031E		BNEQ	59\$...	
		8F	DD	00320	58\$:	PUSHL	#163859		...	
		22	11	00326		BRB	63\$...	
	09	50	91	00328	59\$:	CMPB	R0	#9	...	4019
		05	13	0032B		BEQL	60\$...	
	0B	50	91	0032D		CMPB	R0	#11	...	
		08	12	00330		BNEQ	61\$...	
		8F	DD	00332	60\$:	PUSHL	#163891		...	
		10	11	00338		BRB	63\$...	
	0E	50	91	0033A	61\$:	CMPB	R0	#14	...	4021
		05	13	0033D		BEQL	62\$...	
	7E	01	CE	0033F		MNEGL	#1	-(SP)	...	
		06	11	00342		BRB	63\$...	
		8F	DD	00344	62\$:	PUSHL	#163899		...	
	69	01	FB	0034A	63\$:	CALLS	#1, LIB\$SIGNAL		...	4014
	50	01	DD	0034D	64\$:	MOVL	#1, R0		...	4029
		04	00350		RET				...	4030

; Routine Size: 849 bytes, Routine Base: DBG\$CODE + 15E8


```
3909 4031 1 GLOBAL ROUTINE DBG$EVENT_CANCEL_ALL: NOVALUE =
3910 4032 1
3911 4033 1 FUNCTION
3912 4034 1     This routine cancels all eventpoints.
3913 4035 1
3914 4036 1 INPUTS
3915 4037 1     NONE
3916 4038 1
3917 4039 1 OUTPUTS
3918 4040 1     All command event entries are deleted from the command queue.
3919 4041 1
3920 4042 1
3921 4043 2 BEGIN
3922 4044 2 LOCAL
3923 4045 2     EVENT_ENTRY : REF EVENT$EVENT_DESCRIPTOR; ! Event entry pointer
3924 4046 2
3925 4047 2
3926 4048 2
3927 4049 2 ! Starting at the head of the command queue, 'delete' the command
3928 4050 2 ! entries.
3929 4051 2
3930 4052 2 EVENT_ENTRY = .EVENT$CMD_QUEUE [L_QUEUE_FLINK];
3931 4053 2 WHILE .EVENT_ENTRY NEQA EVENT$CMD_QUEUE DO
3932 4054 2 BEGIN
3933 4055 2
3934 4056 2
3935 4057 2 ! Ignore 'deleted' entries that haven't been removed.
3936 4058 2
3937 4059 2 IF NOT .EVENT_ENTRY [EVENT$V_DELETED]
3938 4060 2 THEN
3939 4061 2
3940 4062 2
3941 4063 2 ! If this is a BREAK, TRACE, or WATCH command, delete it.
3942 4064 2
3943 4065 2 IF (.EVENT_ENTRY [EVENT$B_CMD_TYPE] EQLU
3944 4066 2     EVENT$R_TYPE_BREAK
3945 4067 2 ) OR
3946 4068 2 (.EVENT_ENTRY [EVENT$B_CMD_TYPE] EQLU
3947 4069 2     EVENT$R_TYPE_TRACE
3948 4070 2 ) OR
3949 4071 2 (.EVENT_ENTRY [EVENT$B_CMD_TYPE] EQLU
3950 4072 2     EVENT$R_TYPE_WATCH
3951 4073 2 )
3952 4074 2 THEN
3953 4075 2     EVENT_ENTRY [EVENT$V_DELETED] = TRUE;
3954 4076 2
3955 4077 2
3956 4078 2 ! Point to the next Command Queue entries.
3957 4079 2
3958 4080 2 EVENT_ENTRY = .EVENT_ENTRY [EVENT$L_CMD_FLINK];
3959 4081 2 END;
3960 4082 1 END;
```

DBGEVENT
V04-000

H 14
16-Sep-1984 00:59:10
14-Sep-1984 12:16:53

VAX-11 Bliss-32 V4.0-742
[DEBUG.SRC]DBGEVENT.B32;1

Page 116
(12)

52	00000000'	EF	9E	00002	.ENTRY	DBG\$EVENT_CANCEL_ALL, Save R2	:	4031
50		62	D0	00009	MOVAB	EVENT\$CMD_QUEUE, R2	:	
51		62	9E	0000C	MOVL	EVENT\$CMD_QUEUE, EVENT_ENTRY	:	4052
51		50	D1	0000F	MOVAB	EVENT\$CMD_QUEUE, R1	:	4053
		1F	13	00012	CMPL	EVENT_ENTRY, R1	:	
	17	A0	95	00014	BEQL	4\$:	
		15	19	00017	TSTB	23(EVENT_ENTRY)	:	4059
51	14	A0	9A	00019	BLSS	3\$:	
		0A	13	0001D	MOVZBL	20(EVENT_ENTRY), R1	:	4065
01		51	91	0001F	BEQL	2\$:	
		05	13	00022	CMPB	R1, #1	:	4068
02		51	91	00024	BEQL	2\$:	
		05	12	00027	CMPB	R1, #2	:	4071
17	A0	8F	88	00029	BNEQ	3\$:	
50		60	D0	0002E	BISB2	#128, 23(EVENT_ENTRY)	:	4075
		D9	11	00031	MOVL	(EVENT_ENTRY), EVENT_ENTRY	:	4080
		04	00033	4\$:	BRB	1\$:	4053
					RET		:	4082

; Routine Size: 52 bytes, Routine Base: DBG\$CODE + 1939


```
3962 4083 1 GLOBAL ROUTINE DBG$ACTIVATE_EVENTS: NOVALUE =
3963 4084 1
3964 4085 1 FUNCTION
3965 4086 1     This routine activates all set events.
3966 4087 1
3967 4088 1 INPUTS
3968 4089 1     NONE
3969 4090 1
3970 4091 1 OUTPUTS
3971 4092 1     The events are activated, including the setting of breakpoints, the
3972 4093 1     setting of protections, etc. The exception lists are also built.
3973 4094 1
3974 4095 1
3975 4096 2 BEGIN
3976 4097 2
3977 4098 2 LOCAL
3978 4099 2     USERS_FP: REF BLOCK [,BYTE],
3979 4100 2     EVENT_ENTRY : REF EVENT$EVENT_DESCRIPTOR, ! Event entry pointer
3980 4101 2     NEXT_ENTRY : REF EVENT$EVENT_DESCRIPTOR, ! Event entry pointer
3981 4102 2     PAGE_ENTRY : REF EVENT$PAGE_DESCRIPTOR; ! Page entry pointer
3982 4103 2
3983 4104 2
3984 4105 2 ! Turn off the TBIT flag, which will be set by any event(s) that need
3985 4106 2 ! to use it....
3986 4107 2
3987 4108 2 DBG$RUNFRAME [DBG$V_TBIT] = FALSE;
3988 4109 2
3989 4110 2
3990 4111 2 ! Point to the first Command Queue entry, and begin to process each entry
3991 4112 2 ! in turn.
3992 4113 2
3993 4114 2 USERS_FP = 0;
3994 4115 2 EVENT_ENTRY = .EVENT$CMD_QUEUE [L_QUEUE_FLINK];
3995 4116 2 WHILE .EVENT_ENTRY NEQA EVENT$CMD_QUEUE [A_QUEUE_ENTRY] DO
3996 4117 2 BEGIN
3997 4118 2
3998 4119 2
3999 4120 2 ! Point to the next entry (in case we end up deleting this one !!!
4000 4121 2 !
4001 4122 2 NEXT_ENTRY = .EVENT_ENTRY [EVENT$L_CMD_FLINK];
4002 4123 2
4003 4124 2
4004 4125 2 ! 'Delete' IGNOR entries....
4005 4126 2
4006 4127 2 IF .EVENT_ENTRY [EVENT$B_CMD_TYPE] EQLU EVENT$K_TYPE_IGNORE
4007 4128 2 THEN
4008 4129 2     EVENT_ENTRY [EVENT$V_DELETED] = TRUE;
4009 4130 2
4010 4131 2
4011 4132 2 ! Delete or Activate this event.
4012 4133 2 !
4013 4134 2 IF .EVENT_ENTRY [EVENT$V_DELETED]
4014 4135 2 THEN
4015 4136 2 BEGIN
4016 4137 2
4017 4138 2 ! *****SSI
4018 4139 2 ! EVENT$K_TYPE_SSINT Event is put on Event List, when we see
```



```
: 4019      4140  4      ! DBG$SS_INT signal comes in (DEBUG interception routine is
: 4020      4141  4      ! called, triggered by System Service Call). Bit 15 is set in
: 4021      4142  4      ! saved PSW in the FP for System service. When system service
: 4022      4143  4      ! returns, DEBUG catches Reserved Operand Fault, EVENT$K_TYPE_SSINT
: 4023      4144  4      ! is marked deleted. Now, we just about to take this event entry
: 4024      4145  4      ! off the Event List.
: 4025      4146  4      ! Check to see if we just got back from Reserved Operand Fault
: 4026      4147  4      ! from intercept system service RET, if it is, remember FP.
: 4027      4148  4      ! For we need to turn S/RET active entry to delete entry.
: 4028      4149  4      ! S/RET was built from step over the CALLS SYS$XXX entry, since
: 4029      4150  4      ! we already got Reserved Operand Fault from RET, so we don't
: 4030      4151  4      ! want to do it again, so later on we can turn S/RET into normal
: 4031      4152  4      ! step entry.
: 4032      4153  4
: 4033      4154  4      IF .EVENT_ENTRY [EVENT$B_CMD_TYPE] EQL EVENT$K_TYPE_SSINT
: 4034      4155  4      THEN
: 4035      4156  4          USERS_FP = .EVENT_ENTRY [EVENT$L_USERS_FP]
: 4036      4157  4      ELSE
: 4037      4158  4          USERS_FP = 0;
: 4038      4159  4      DELETE_EVENT_ENTRY (.EVENT_ENTRY);
: 4039      4160  4      END
: 4040      4161  3      ELSE
: 4041      4162  3          DBG$ACTIVATE_EVENT (.EVENT_ENTRY);
: 4042      4163  3
: 4043      4164  3
: 4044      4165  3      ! Point to the next event and continue.
: 4045      4166  3      !
: 4046      4167  3      EVENT_ENTRY = .NEXT_ENTRY;
: 4047      4168  3
: 4048      4169  3
: 4049      4170  3      ! *****SSI
: 4050      4171  3      ! Check to see if this the case we noted above. Turn S/RET into
: 4051      4172  3      ! normal Step entry.
: 4052      4173  3      !
: 4053      4174  3      IF .USERS_FP NEQ 0
: 4054      4175  3      THEN
: 4055      4176  4          BEGIN
: 4056      4177  4              IF .EVENT_ENTRY [EVENT$B_SUB_KIND] EQL EVENT$K_ACC_RTRN
: 4057      4178  4              THEN
: 4058      4179  5                  BEGIN
: 4059      4180  5                      IF .EVENT_ENTRY [EVENT$B_CMD_TYPE] EQL EVENT$K_TYPE_STEPS
: 4060      4181  5                      THEN
: 4061      4182  5                          IF .USERS_FP EQL .EVENT_ENTRY [EVENT$L_USERS_FP]
: 4062      4183  5                          THEN
: 4063      4184  5
: 4064      4185  5
: 4065      4186  5                      ! Change it to regular step entry.
: 4066      4187  5                      !
: 4067      4188  6                      BEGIN
: 4068      4189  6                          IF .EVENT_ENTRY [EVENT$V_STEP_BPT]
: 4069      4190  6                          THEN
: 4070      4191  7                              BEGIN
: 4071      4192  7                                  EVENT_ENTRY [EVENT$V_STEP_BPT] = FALSE;
: 4072      4193  7                                  EVENT_ENTRY [EVENT$B_CMD_KIND] = EVENT$K_KIND_INS;
: 4073      4194  7                                  EVENT_ENTRY [EVENT$B_SUB_KIND] = .EVENT_ENTRY [EVENT$B_SAVED_SUB_KIND];
: 4074      4195  6                                  END;
: 4075      4196  5                              END;
:                                     END;
```



```

: 4076      4197      4
: 4077      4198      3
: 4078      4199      2
: 4079      4200      2
: 4080      4201      2
: 4081      4202      2
: 4082      4203      2
: 4083      4204      2
: 4084      4205      2
: 4085      4206      2
: 4086      4207      2
: 4087      4208      2
: 4088      4209      2
: 4089      4210      2
: 4090      4211      2
: 4091      4212      2
: 4092      4213      2
: 4093      4214      2
: 4094      4215      2
: 4095      4216      3
: 4096      4217      3
: 4097      4218      3
: 4098      4219      4
: 4099      4220      4
: 4100      4221      4
: 4101      4222      4
: 4102      4223      4
: 4103      4224      4
: 4104      4225      4
: 4105      4226      4
: 4106      4227      4
: 4107      4228      5
: 4108      4229      4
: 4109      4230      4
: 4110      4231      4
: 4111      4232      4
: 4112      4233      3
: 4113      4234      2
: 4114      4235      2
: 4115      4236      2
: 4116      4237      2
: 4117      4238      2
: 4118      4239      2
: 4119      4240      2
: 4120      4241      2
: 4121      4242      2
: 4122      4243      3
: 4123      4244      3
: 4124      4245      3
: 4125      4246      3
: 4126      4247      3
: 4127      4248      3
: 4128      4249      3
: 4129      4250      2
: 4130      4251      2
: 4131      4252      1

      END;
END;

      As long as we were not skipping ACCVIO's, for each entry in the page
      queue, write protect the page described. If we ARE skipping then
      we have to TBIT.
      *****SSI
      If we are in the middle of the system service cycle, we don't want
      want to put the protection back, for if we do, we may cause the
      system service to fail again. Note: Some system service
      calls system service, the cycle starts from the call in user program
      till this call is returned to the user program. During this time
      frame, we don't want to put protection back.

      IF NOT .SKIP_ACCVIOS AND
      NOT .DBG$GB_SET_WATCH_FLAG
      THEN
      BEGIN
      PAGE_ENTRY = .EVENT$PAGE_QUEUE [L_QUEUE_FLINK];
      WHILE .PAGE_ENTRY NEQA EVENT$PAGE_QUEUE DO
      BEGIN
      LOCAL
      ADDRESS :      VECTOR [2];      ! Address range for protect

      ADDRESS [0] = .PAGE_ENTRY [EVENT$L_PAGE_ADDRESS];
      ADDRESS [1] = .PAGE_ENTRY [EVENT$L_PAGE_ADDRESS];
      IF NOT $SETPRT (INADR      = ADDRESS,
      PROT      = PRT$C_UR,
      PRVPRT      = PAGE_ENTRY [EVENT$B_PAGE_PROTECTION]
      )
      THEN
      SIGNAL (DBG$_NOWPROT)
      ELSE
      PAGE_ENTRY = .PAGE_ENTRY [EVENT$L_PAGE_FLINK];
      END;
      END;

      ! If we are skipping ACCVIO and WATCHES, T-bit over the cause.
      ! (just had a accvio from the instruction write to the page that was
      ! protected for watch pointing) or (just had a RET from System Service).

      IF .SKIP_ACCVIOS OR .SKIP_WATCHES
      THEN
      BEGIN
      ! About to TBIT over the instruction that triggered a watchpoint.
      ! We also have to make sure that we don't get interrupted by ASTs.
      DBG$RUNFRAME [DBG$V_TBIT] = TRUE;
      DBG$GV_CONTROL[DBG$V_CONTROL_TBIT] = TRUE;
      END;
      END;
```


			00FC 00000	.ENTRY	DBG\$ACTIVATE_EVENTS, Save R2,R3,R4,R5,R6,R7	4083
57	000000000G	00	9E 00002	MOVAB	DBG\$RUNFRAME+68, R7	
56	000000000'	EF	9E 00009	MOVAB	SKIP ACCVIOS, R6	
55	000000000'	EF	9E 00010	MOVAB	EVENT\$CMD_QUEUE, R5	
5E		08	C2 00017	SUBL2	#8, SP	
67		10	8A 0001A	BICB2	#16, DBG\$RUNFRAME+68	4108
		53	D4 0001D	CLRL	USERS_FP	4114
52		65	D0 0001F	MOVL	EVENT\$CMD_QUEUE, EVENT_ENTRY	4115
50		65	9E 00022	MOVAB	EVENT\$CMD_QUEUE, R0	4116
50		52	D1 00025	CMPL	EVENT_ENTRY, R0	
		5E	13 00028	BEQL	7\$	
54		62	D0 0002A	MOVL	(EVENT_ENTRY), NEXT_ENTRY	4122
05	14	A2	91 0002D	CMPB	20(EVENT_ENTRY), #5	4127
		05	12 00031	BNEQ	2\$	
17	A2	80	8F 88 00033	BISB2	#128, 23(EVENT_ENTRY)	4129
		17	A2 95 00038	TSTB	23(EVENT_ENTRY)	4134
		17	18 0003B	BGEQ	5\$	
06	14	A2	91 0003D	CMPB	20(EVENT_ENTRY), #6	4154
		06	12 00041	BNEQ	3\$	
53	24	A2	D0 00043	MOVL	36(EVENT_ENTRY), USERS_FP	4156
		02	11 00047	BRB	4\$	
		53	D4 00049	CLRL	USERS_FP	4158
		52	DD 0004B	PUSHL	EVENT_ENTRY	4159
0000V	CF	01	FB 0004D	CALLS	#1, DELETE_EVENT_ENTRY	
		07	11 00052	BRB	6\$	4134
		52	DD 00054	PUSHL	EVENT_ENTRY	4162
0000V	CF	01	FB 00056	CALLS	#1, DBG\$ACTIVATE_EVENT	
	52	54	D0 0005B	MOVL	NEXT_ENTRY, EVENT_ENTRY	4167
		53	D5 0005E	TSTL	USERS_FP	4174
		C0	13 00060	BEQL	1\$	
	04	A2	91 00062	CMPB	22(EVENT_ENTRY), #4	4177
		BA	12 00066	BNEQ	1\$	
	03	A2	91 00068	CMPB	20(EVENT_ENTRY), #3	4180
		B4	12 0006C	BNEQ	1\$	
24	A2	53	D1 0006E	CMPL	USERS_FP, 36(EVENT_ENTRY)	4182
		AE	12 00072	BNEQ	1\$	
A9	19	A2	02 E1 00074	BBC	#2, 25(EVENT_ENTRY), 1\$	4189
	19	A2	04 8A 00079	BICB2	#4, 25(EVENT_ENTRY)	4192
	15	A2	01 90 0007D	MOVB	#1, 21(EVENT_ENTRY)	4193
	16	A2	A2 90 00081	MOVB	27(EVENT_ENTRY), 22(EVENT_ENTRY)	4194
		9A	11 00086	BRB	1\$	4116
	49	66	E8 00088	BLBS	SKIP ACCVIOS, 11\$	4213
	3E	FD	A5 E8 0008B	BLBS	DBG\$GB SET_WATCH_FLAG, 10\$	4214
	52	30	A5 D0 0008F	MOVL	EVENT\$PAGE_QUEUE, PAGE_ENTRY	4217
	50	30	A5 9E 00093	MOVAB	EVENT\$PAGE_QUEUE, R0	4218
	50	52	D1 00097	CMPL	PAGE_ENTRY, R0	
		31	13 0009A	BEQL	10\$	
	6E	08	A2 D0 0009C	MOVL	8(PAGE_ENTRY), ADDRESS	4223
04	AE	08	A2 D0 000A0	MOVL	8(PAGE_ENTRY), ADDRESS+4	4224
		0E	A2 9F 000A5	PUSHAB	14(PAGE_ENTRY)	4228
		0F	DD 000A8	PUSHL	#15	
		7E	7C 000AA	CLRQ	-(SP)	
		10	AE 9F 000AC	PUSHAB	ADDRESS	

DBGEVENT
V04-000

M 14
16-Sep-1984 00:59:10
14-Sep-1984 12:16:53

VAX-11 Bliss-32 V4.0-742
[DEBUG.SRC]DBGEVENT.B32;1

Page 121
(13)

00000000G	00		05	FB	000AF	CALLS	#5, SYSS\$SETPRT	:
	0F		50	E8	000B6	BLBS	R0, 9\$:
		000284C4	8F	DD	000B9	PUSHL	#165060	: 4230
00000000G	00		01	FB	000BF	CALLS	#1, LIB\$SIGNAL	:
			CB	11	000C6	BRB	8\$:
	52		62	D0	000C8	9\$:	MOVL (PAGE_ENTRY), PAGE_ENTRY	: 4232
			C6	11	000CB	BRB	8\$: 4218
	04		66	E8	000CD	10\$:	BLBS SKIP_ACCVIOS, 11\$: 4241
	0A	04	A6	E9	000D0	BLBC	SKIP_WATCHES, 12\$:
	67		10	88	000D4	11\$:	BISB2 #16, DBG\$RUNFRAME+68	: 4248
00000000G	00		04	88	000D7	11\$:	BISB2 #4, DBG\$GV_CONTROL+1	: 4249
			04	000DE	12\$:	RET		: 4252

; Routine Size: 223 bytes, Routine Base: DBG\$CODE + 196D

```
4133 4253 1 GLOBAL ROUTINE DBGSACTIVATE_EVENT (EVENT): NOVALUE =
4134 4254 1
4135 4255 1 FUNCTION
4136 4256 1     This routine activates an event. Included in this action is the
4137 4257 1     building of a linked list for each exception type used to implement
4138 4258 1     any event: BPT, TBIT, and ACCVIO.
4139 4259 1
4140 4260 1 INPUTS
4141 4261 1     EVENT :          Pointer to an event entry.
4142 4262 1
4143 4263 1 OUTPUTS
4144 4264 1     The event is activated, including the setting of breakpoints, the
4145 4265 1     setting of protections, etc. The exception lists are also built.
4146 4266 1
4147 4267 1
4148 4268 2 BEGIN
4149 4269 2 MAP EVENT :          REF EVENT$EVENT_DESCRIPTOR; ! Event entry pointer
4150 4270 2 BUILTIN PROBER;
4151 4271 2
4152 4272 2 LOCAL
4153 4273 2     ADDRESS,
4154 4274 2     AT_LIBSIGNAL,
4155 4275 2     CALL_FRAME: REF VECTOR[.LONG],
4156 4276 2     IGNORE_ENTRY : REF EVENT$EVENT_DESCRIPTOR,
4157 4277 2     EVENT_ADDRESS : REF VECTOR[.BYTE],
4158 4278 2     USER$FP :      REF BLOCK[.BYTE],
4159 4279 2     RET_PC,
4160 4280 2     STATUS,
4161 4281 2     USER$PC: REF VECTOR[.BYTE];
4162 4282 2
4163 4283 2 ! Case on the kind of event entry, to determine whether how to activate
4164 4284 2 ! the entry.
4165 4285 2
4166 4286 2 CASE .EVENT [EVENT$B_CMD_KIND]
4167 4287 2 FROM EVENT$K_KIND_ACC TO EVENT$K_KIND_EXC OF
4168 4288 2 SET
4169 4289 2 [EVENT$K_KIND_ACC]:
4170 4290 2
4171 4291 2     ! Case on the access sub-kind....
4172 4292 2
4173 4293 2 CASE .EVENT[EVENT$B_SUB_KIND]
4174 4294 2 FROM EVENT$K_ACC_READ TO EVENT$K_ACC_RTRN OF
4175 4295 2 SET
4176 4296 2
4177 4297 2 [EVENT$K_ACC_READ]:
4178 4298 2 ;
4179 4299 2
4180 4300 2 [EVENT$K_ACC_WRITE]:
4181 4301 2 ;
4182 4302 2
4183 4303 2 [EVENT$K_ACC_MDFY]:
4184 4304 2 ;
4185 4305 2
4186 4306 2
4187 4307 2
4188 4308 2
4189 4309 2
```


4190	4310	2
4191	4311	2
4192	4312	2
4193	4313	2
4194	4314	2
4195	4315	2
4196	4316	2
4197	4317	2
4198	4318	2
4199	4319	2
4200	4320	2
4201	4321	2
4202	4322	2
4203	4323	2
4204	4324	2
4205	4325	2
4206	4326	2
4207	4327	2
4208	4328	2
4209	4329	2
4210	4330	2
4211	4331	2
4212	4332	2
4213	4333	2
4214	4334	2
4215	4335	4
4216	4336	4
4217	4337	4
4218	4338	4
4219	4339	4
4220	4340	4
4221	4341	4
4222	4342	4
4223	4343	4
4224	4344	4
4225	4345	4
4226	4346	4
4227	4347	4
4228	4348	4
4229	4349	4
4230	4350	4
4231	4351	4
4232	4352	4
4233	4353	4
4234	4354	4
4235	4355	4
4236	4356	4
4237	4357	4
4238	4358	4
4239	4359	4
4240	4360	4
4241	4361	4
4242	4362	4
4243	4363	4
4244	4364	4
4245	4365	4
4246	4366	3

! Execution Access means that we've got to put a BPT
! instruction at the address indicated.

[EVENT\$K_ACC_EXEC]:
BEGIN

! Get the event's address.

EVENT_ADDRESS = .EVENT [EVENT\$L_ADDRESS];

! Next, copy the opcode at the event's address into the
! event entry for safe keeping.

EVENT [EVENT\$B_OPCODE] = .EVENT_ADDRESS [0];

! If the address we're to place a BPT is the current user
! address, (meaning we're about to put a BPT where we're
! about to start), create a ignore-event instead....

IF .EVENT_ADDRESS EQ LA .DBG\$RUNFRAME[DBG\$L_USER_PC]
THEN
BEGIN

! Get a new event descriptor, and link it into the
! command queue. Note that this entry is placed on
! the HEAD of the queue, and should be encountered
! first !!!

IGNOR_ENTRY = DBG\$GET_MEMORY (EVENT\$K_EVENT_DESCRIPTOR_SIZE);
INSQUE (.IGNOR_ENTRY, EVENT\$CMD_QUEUE);

! Setup this entry to be a ignore-event. Note that
! we don't really need to initialize more than the
! CMD_TYPE, including linking to the event entry
! causing this mess, 'cause this entry will just be
! deleted. However, maybe in the future....

IGNOR_ENTRY [EVENT\$B_CMD_TYPE] = EVENT\$K_TYPE_IGNORE;
IGNOR_ENTRY [EVENT\$B_CMD_KIND] = EVENT\$K_KIND_ACC;
IGNOR_ENTRY [EVENT\$B_SUB_KIND] = EVENT\$K_ACC_EXEC;
IGNOR_ENTRY [EVENT\$V_ONCE_ONLY] = TRUE;
INSQUE1 (.IGNOR_ENTRY, .EVENT);

! Turn on the TBIT flag so that we'll TBIT over the
! next instruction, and return.

DBG\$RUNFRAME [DBG\$V_TBIT] = TRUE;
DBG\$GV_CONTROL[DBG\$V_CONTROL_TBIT] = TRUE;
RETURN;
END;

4247	4367	3
4248	4368	3
4249	4369	3
4250	4370	3
4251	4371	3
4252	4372	3
4253	4373	3
4254	4374	3
4255	4375	4
4256	4376	3
4257	4377	4
4258	4378	4
4259	4379	4
4260	4380	3
4261	4381	2
4262	4382	2
4263	4383	2
4264	4384	2
4265	4385	2
4266	4386	2
4267	4387	2
4268	4388	3
4269	4389	3
4270	4390	3
4271	4391	3
4272	4392	3
4273	4393	3
4274	4394	3
4275	4395	3
4276	4396	3
4277	4397	4
4278	4398	4
4279	4399	4
4280	4400	4
4281	4401	4
4282	4402	4
4283	4403	4
4284	4404	4
4285	4405	4
4286	4406	5
4287	4407	5
4288	4408	5
4289	4409	5
4290	4410	5
4291	4411	5
4292	4412	4
4293	4413	4
4294	4414	4
4295	4415	4
4296	4416	4
4297	4417	4
4298	4418	4
4299	4419	4
4300	4420	4
4301	4421	4
4302	4422	4
4303	4423	4

```
! BPT is okay, write a BPT instruction into that address.
! Note that we won't actually do this if the original
! opcode was a BPT.
IF .EVENT [EVENT$B_OPCODE] NEQU BPT_INSTRUCTION
THEN
  IF NOT WRITE_OPCODE (.EVENT_ADDRESS, BPT_INSTRUCTION)
  THEN
    BEGIN
      SIGNAL (DBG$NOWBPT);
      RETURN;
    END;
END;

! Return Access means that we've got to put a BPT
! instruction at the address indicated.
[EVENT$K_ACC_RTRN]:
BEGIN

  ! Depending on the command type (STEP or otherwise)...
  IF .EVENT [EVENT$B_CMD_TYPE] EQLU EVENT$K_TYPE_STEPS OR
    .EVENT [EVENT$B_CMD_TYPE] EQLU EVENT$K_TYPE_SKIPS OR
    .EVENT [EVENT$B_CMD_TYPE] EQLU EVENT$K_TYPE_SSINT
  THEN
    BEGIN

      ! If the save FP is 0, then we need to initialize
      ! with the current one.
      USERS_FP = .EVENT [EVENT$L_USERS_FP];
      IF .USERS_FP EQLA 0
      THEN
        BEGIN
          USERS_FP = .DBG$RUNFRAME [DBG$L_USER_FP];
          EVENT [EVENT$L_USERS_FP] = .USERS_FP;
          CALL_FRAME = DBG$GET_MEMORY(5);
          EVENT [EVENT$L_CALL_FRAME] = .CALL_FRAME;
          CH$MOVE(20, .EVENT [EVENT$L_USERS_FP], .CALL_FRAME);
        END;

      IF NOT PROBER (%REF(0), %REF(20), .USERS_FP)
      THEN
        RETURN;

      ! Check for the special case where we have just
      ! called LIB$SIGNAL or LIB$STOP. The flag
      ! AT_LIB$SIGNAL will be set to true if we are
      ! at LIB$SIGNAL or LIB$STOP
```



```
: 4304      4424  4
: 4305      4425  4
: 4306      4426  4
: 4307      4427  4
: 4308      4428  4
: 4309      4429  4
: 4310      4430  4
: 4311      4431  4
: 4312      4432  4
: 4313      4433  4
: 4314      4434  5
: 4315      4435  4
: 4316      4436  4
: 4317      4437  4
: 4318      4438  4
: 4319      4439  4
: 4320      4440  4
: 4321      4441  4
: 4322      4442  4
: 4323      4443  4
: 4324      4444  4
: 4325      4445  4
: 4326      4446  4
: 4327      4447  5
: 4328      4448  4
: 4329      4449  4
: 4330      4450  4
: 4331      4451  4
: 4332      4452  4
: 4333      4453  4
: 4334      4454  4
: 4335      4455  4
: 4336      4456  4
: 4337      4457  4
: 4338      4458  4
: 4339      4459  5
: 4340      4460  5
: 4341      4461  6
: 4342      4462  5
: 4343      4463  6
: 4344      4464  6
: 4345      4465  6
: 4346      4466  6
: 4347      4467  6
: 4348      4468  6
: 4349      4469  6
: 4350      4470  6
: 4351      4471  6
: 4352      4472  6
: 4353      4473  6
: 4354      4474  6
: 4355      4475  6
: 4356      4476  6
: 4357      4477  6
: 4358      4478  7
: 4359      4479  6
: 4360      4480  6
```

```
USERS_PC = .DBG$RUNFRAME[DBG$L_USER_PC];
AT_LIBSIGNAL = FALSE;
```

```
! Check for current PC being at one of these
! locations. Here the locations we are checking
! are the addresses of LIB$SIGNAL and LIB$STOP
! in the LIBRTL shareable library.
```

```
IF (.USERS_PC EQL LIB$SIGNAL+2) OR
   (.USERS_PC EQL LIB$STOP+2)
THEN
    AT_LIBSIGNAL = TRUE;
```

```
! Also check for the current PC being at the
! address of LIB$SIGNAL or LIB$STOP if these
! were linked into the users program with
! /NOSYSSHR on the link. In this case the
! symbols show up in the GST and we remember
! their addresses when we initialize the GST.
```

```
IF (.USERS_PC EQL .DBG$GL_LIB_SIGNAL_ADDR+2) OR
   (.USERS_PC EQL .DBG$GL_LIB_STOP_ADDR+2)
THEN
    AT_LIBSIGNAL = TRUE;
```

```
! Also check for instruction at current location
! being a JMP to LIB$SIGNAL or LIB$STOP.
```

```
IF PROBER(%REF(0),
          %REF(6),
          .USERS_PC)
```

```
THEN
    BEGIN
        IF (.USERS_PC[0] EQL %X'17') AND
           (.USERS_PC[1] EQL %X'FF')
        THEN
            BEGIN
```

```
! Get the jump address by adding the PC
! value plus the 6 bytes for the
! JMP instruction itself.
```

```
CH$MOVE(4, USERS_PC[2], ADDRESS);
ADDRESS = .ADDRESS + .USERS_PC + 6;
IF PROBER(%REF(0),
          %REF(4),
          .ADDRESS)
```

```
THEN
    ADDRESS = ..ADDRESS;
```

```
IF (.ADDRESS EQL LIB$SIGNAL+2) OR
   (.ADDRESS EQL LIB$STOP+2)
```

```
THEN
    AT_LIBSIGNAL = TRUE;
```



```
: 4361      4481  5
: 4362      4482  4
: 4363      4483  4
: 4364      4484  4
: 4365      4485  4
: 4366      4486  5
: 4367      4487  5
: 4368      4488  5
: 4369      4489  5
: 4370      4490  5
: 4371      4491  5
: 4372      4492  5
: 4373      4493  5
: 4374      4494  5
: 4375      4495  5
: 4376      4496  5
: 4377      4497  5
: 4378      4498  5
: 4379      4499  5
: 4380      4500  5
: 4381      4501  5
: 4382      4502  5
: 4383      4503  6
: 4384      4504  5
: 4385      4505  5
: 4386      4506  5
: 4387      4507  4
: 4388      4508  4
: 4389      4509  4
: 4390      4510  4
: 4391      4511  4
: 4392      4512  4
: 4393      4513  4
: 4394      4514  4
: 4395      4515  4
: 4396      4516  4
: 4397      4517  4
: 4398      4518  4
: 4399      4519  4
: 4400      4520  4
: 4401      4521  4
: 4402      4522  3
: 4403      4523  4
: 4404      4524  4
: 4405      4525  4
: 4406      4526  4
: 4407      4527  4
: 4408      4528  4
: 4409      4529  4
: 4410      4530  4
: 4411      4531  4
: 4412      4532  4
: 4413      4533  4
: 4414      4534  4
: 4415      4535  4
: 4416      4536  4
: 4417      4537  4
```

```
END;
END;
IF .AT_LIBSIGNAL
THEN
BEGIN
! In this case, we do not want want to use
! the dirty-the-tbit approach because
! LIB$SIGNAL and LIB$STOP mess with the stack.
! So we revert to the temporary breakpoint
! method.
RET_PC = .USERS_FP[SF$SAVE_PC];
EVENT [EVENT$V_STEP_BPT] = TRUE;
EVENT [EVENT$B_CMD_KIND] = EVENT$K_KIND_INS;
EVENT [EVENT$B_SUB_KIND] = .EVENT [EVENT$B_SAVED_SUB_KIND];
IF .EVENT [EVENT$B_SUB_KIND] EQL 0
THEN
EVENT [EVENT$B_SUB_KIND] = EVENT$K_INS_EVR;
EVENT [EVENT$L_ADDRESS] = .RET_PC;
EVENT [EVENT$B_OPCODE] = .(.RET_PC)<0,8>;
IF NOT WRITE_OPCODE (.RET_PC, BPT_INSTRUCTION)
THEN
SIGNAL(DBG$_NOWBPT);
RETURN;
END;

! Turn on bit 15 in the user's run-
! frame to get a reserved-operand
! fault on the return.
MATCH_RIGHT_CALL_FRAME(.EVENT);
USERS_FP = .EVENT [EVENT$L_USERS_FP];
(USERS_FP [SF$W_SAVE_PSW])<15,1,0> = TRUE;
END

! This is a [BREAK:TRACE]/RETURN to set up.
ELSE
BEGIN

! Get the event's address.
EVENT_ADDRESS = .EVENT [EVENT$L_ADDRESS];

! Next, copy the opcode at the event's address into the
! event entry for safe keeping.
EVENT [EVENT$B_OPCODE] = .EVENT_ADDRESS [0];

! If the address we're to place a BPT is the current
```



```

: 4418      4538      4
: 4419      4539      4
: 4420      4540      4
: 4421      4541      4
: 4422      4542      4
: 4423      4543      4
: 4424      4544      5
: 4425      4545      5
: 4426      4546      5
: 4427      4547      5
: 4428      4548      5
: 4429      4549      5
: 4430      4550      5
: 4431      4551      5
: 4432      4552      5
: 4433      4553      5
: 4434      4554      5
: 4435      4555      5
: 4436      4556      5
: 4437      4557      5
: 4438      4558      5
: 4439      4559      5
: 4440      4560      5
: 4441      4561      5
: 4442      4562      5
: 4443      4563      5
: 4444      4564      5
: 4445      4565      5
: 4446      4566      5
: 4447      4567      5
: 4448      4568      5
: 4449      4569      5
: 4450      4570      5
: 4451      4571      5
: 4452      4572      5
: 4453      4573      5
: 4454      4574      5
: 4455      4575      5
: 4456      4576      5
: 4457      4577      5
: 4458      4578      5
: 4459      4579      5
: 4460      4580      5
: 4461      4581      5
: 4462      4582      6
: 4463      4583      6
: 4464      4584      6
: 4465      4585      5
: 4466      4586      5
: 4467      4587      4
: 4468      4588      4
: 4469      4589      4
: 4470      4590      4
: 4471      4591      4
: 4472      4592      4
: 4473      4593      4
: 4474      4594      4

```

```

: user address, (meaning we're about to put a BPT
: where we're about to start), create a ignore-event
: instead....
IF .EVENT_ADDRESS EQLA .DBG$RUNFRAME[DBG$USER_PC]
THEN
    BEGIN

        : Get a new event descriptor, and link it into the
        : command queue. Note that this entry is placed on
        : the HEAD of the queue, and should be encountered
        : first !!!
        IGNOR_ENTRY = DBG$GET_MEMORY
                        (EVENT$K_EVENT_DESCRIPTOR_SIZE);
        INSQUE (.IGNOR_ENTRY, EVENT$CMD_QUEUE);

        : Setup this entry to be a ignore-event. Note that
        : we don't really need to initialize more than the
        : CMD_TYPE, including linking to the event entry
        : causing this mess, 'cause this entry will just be
        : deleted. However, maybe in the future....
        : If we are at /RET .PC, then we setup a skip entry
        : instead of ignore entry.
        IF .EVENT [EVENT$V_RET_AT_PC]
        THEN
            IGNOR_ENTRY [EVENT$B_CMD_TYPE] = EVENT$K_TYPE_SKIPS
        ELSE
            IGNOR_ENTRY [EVENT$B_CMD_TYPE] = EVENT$K_TYPE_IGNORE;

        IGNOR_ENTRY [EVENT$B_CMD_KIND] = EVENT$K_KIND_ACC;
        IGNOR_ENTRY [EVENT$B_SUB_KIND] = EVENT$K_ACC_RTRN;
        IGNOR_ENTRY [EVENT$V_ONCE_ONLY] = TRUE;
        INSQUE1 (.IGNOR_ENTRY, .EVENT);

        : Turn on the TBIT flag so that we'll TBIT over the
        : next instruction, and return.
        IF NOT .EVENT [EVENT$V_RET_AT_PC]
        THEN
            BEGIN
                DBG$RUNFRAME [DBG$V_TBIT] = TRUE;
                DBG$GV_CONTROL[DBG$V_CONTROL_TBIT] = TRUE;
            END;
        RETURN;
        END;

        : Write a BPT instruction into that address. Note that
        : we won't actually do this if the original opcode was
        : a BPT.
        IF .EVENT [EVENT$B_OPCODE] NEQU BPT_INSTRUCTION

```



```
: 4475 4595 4
: 4476 4596 5
: 4477 4597 4
: 4478 4598 5
: 4479 4599 5
: 4480 4600 5
: 4481 4601 4
: 4482 4602 3
: 4483 4603 2
: 4484 4604 2
: 4485 4605 2
: 4486 4606 2
: 4487 4607 2
: 4488 4608 3
: 4489 4609 3
: 4490 4610 3
: 4491 4611 3
: 4492 4612 3
: 4493 4613 3
: 4494 4614 3
: 4495 4615 4
: 4496 4616 3
: 4497 4617 4
: 4498 4618 4
: 4499 4619 4
: 4500 4620 4
: 4501 4621 4
: 4502 4622 4
: 4503 4623 5
: 4504 4624 6
: 4505 4625 5
: 4506 4626 6
: 4507 4627 6
: 4508 4628 6
: 4509 4629 6
: 4510 4630 6
: 4511 4631 6
: 4512 4632 6
: 4513 4633 6
: 4514 4634 6
: 4515 4635 6
: 4516 4636 6
: 4517 4637 6
: 4518 4638 6
: 4519 4639 6
: 4520 4640 6
: 4521 4641 6
: 4522 4642 6
: 4523 4643 6
: 4524 4644 6
: 4525 4645 5
: 4526 4646 6
: 4527 4647 6
: 4528 4648 6
: 4529 4649 6
: 4530 4650 6
: 4531 4651 6
```

```
THEN
  IF NOT WRITE_OPCODE (.EVENT_ADDRESS, BPT_INSTRUCTION)
  THEN
    BEGIN
      SIGNAL (DBG$_NOWBPT);
      RETURN;
    END;
  END;
TES;
[EVENT$K_KIND_INS]:
BEGIN
  LOCAL OPCODE;
  OPCODE = (.DBG$RUNFRAME[DBG$_USER_PC])<0,8,0>;
  ! If stepping over and a) it's time to BPT or b) we WERE BPTing...
  !
  IF .EVENT [EVENT$V_STEP_OVER] AND
    (.EVENT [EVENT$V_STEP_BPT] OR .DBG$OPCODES_STEP_OVER[.OPCODE])
  THEN
    BEGIN
      ! ... and we're not already in the middle...
      !
      IF NOT .EVENT [EVENT$V_STEP_BPT]
      THEN
        BEGIN
          IF (.OPCODE EQL %X'FA') OR (.OPCODE EQL %X'FB')
          THEN
            BEGIN
              ! This is a CALLS or CALLG instruction. We implement
              ! STEP/OVER by a STEP/INTO,STEP/RETURN,STEP sequence
              ! We do this by temporarily turning the STEP event descriptor into a RETURN
              ! event descriptor (KIND = ACC, SUB_KIND = RTRN). This causes us to dirty up
              ! the call stack and get a reserved operand fault at the return. When we handle the
              ! ROPRAND then we change the return event back into a step event. We save the origin
              ! sub-kind field to enable us to do this correctly.
              !
              EVENT [EVENT$V_STEP_BPT] = TRUE;
              EVENT [EVENT$V_USERS_FP] = 0;
              EVENT [EVENT$B_CMD_KIND] = EVENT$K_KIND_ACC;
              EVENT [EVENT$B_SAVED_SUB_KIND] = .EVENT [EVENT$B_SUB_KIND];
              EVENT [EVENT$B_SUB_KIND] = EVENT$K_ACC_RTRN;
              DBG$RUNFRAME [DBG$_TBIT] = TRUE;
              DBG$GV_CONTROL[DBG$_CONTROL_TBIT] = TRUE;
              RETURN;
            END
          ELSE
            BEGIN
              ! This is a JSB-type instruction. Get the address
              ! of the next instruction.
              !
              EVENT_ADDRESS = DBG$INS_DECODE (
```



```
: 4532      4652  6
: 4533      4653  6
: 4534      4654  6
: 4535      4655  6
: 4536      4656  6
: 4537      4657  6
: 4538      4658  6
: 4539      4659  6
: 4540      4660  6
: 4541      4661  6
: 4542      4662  6
: 4543      4663  6
: 4544      4664  6
: 4545      4665  7
: 4546      4666  7
: 4547      4667  7
: 4548      4668  7
: 4549      4669  7
: 4550      4670  7
: 4551      4671  7
: 4552      4672  7
: 4553      4673  7
: 4554      4674  7
: 4555      4675  7
: 4556      4676  7
: 4557      4677  7
: 4558      4678  7
: 4559      4679  7
: 4560      4680  7
: 4561      4681  6
: 4562      4682  6
: 4563      4683  6
: 4564      4684  6
: 4565      4685  6
: 4566      4686  6
: 4567      4687  6
: 4568      4688  6
: 4569      4689  6
: 4570      4690  6
: 4571      4691  6
: 4572      4692  6
: 4573      4693  6
: 4574      4694  6
: 4575      4695  5
: 4576      4696  5
: 4577      4697  5
: 4578      4698  5
: 4579      4699  5
: 4580      4700  5
: 4581      4701  5
: 4582      4702  4
: 4583      4703  4
: 4584      4704  4
: 4585      4705  4
: 4586      4706  4
: 4587      4707  4
: 4588      4708  4
```

```
.DBG$RUNFRAME[DBG$L_USER_PC], 0, FALSE);

! If the address is in system or P1 space...
IF .EVENT_ADDRESS GEQA P1_SPACE OR
.EVENT_ADDRESS EQLA 0
THEN

    ! then we're going to TBIT ('cause we won't set a
    ! BPT in P1 space or in system space).
    BEGIN

        ! Check to see if this is address 0 or we're stepping
        ! nosystem. If so - belch.
        IF .EVENT [EVENT$V_STEP_NOSYSTEM] OR
        .EVENT_ADDRESS EQLA 0
        THEN
            SIGNAL (DBG$_STEPINTO, 1, .DBG$RUNFRAME[DBG$L_USER_PC]);

        ! We're TBIT'n now...
        !
        DBG$RUNFRAME [DBG$V_TBIT] = TRUE;
        RETURN;
        END;

        ! Otherwise, everything is hunky dory, so continue. We'll
        ! Set up the address in the event entry for the BPT.
        ! Save the next address in to entry for the step's BPT.
        EVENT [EVENT$L_ADDRESS] = .EVENT_ADDRESS;

        ! Flag that a stepping-into BPT is set.
        EVENT [EVENT$V_STEP_BPT] = TRUE;
        END;
    END

! We're in the middle of stepping over, but we've been
! interrupted, so continue....
ELSE
    EVENT_ADDRESS = .EVENT [EVENT$L_ADDRESS];

! We get here when a) we're in the middle of a step/over (i.e.,
! something interrupted the step/over), or we're at the start
```



```
: 4589      4709  4      | of one (i.e. at a CALL/JSB-type instruction), and we've got
: 4590      4710  4      | the address set up in the event entry (see above).
: 4591      4711  4      |
: 4592      4712  4      | Next, copy the opcode at the event's address into the
: 4593      4713  4      | event entry for safe keeping.
: 4594      4714  4      |
: 4595      4715  4      | EVENT [EVENT$B_OPCODE] = .EVENT_ADDRESS [0];
: 4596      4716  4      |
: 4597      4717  4      |
: 4598      4718  4      | Finally, write a BPT instruction into that address.
: 4599      4719  4      | Note that we won't actually do this if the original
: 4600      4720  4      | opcode was a BPT.
: 4601      4721  4      |
: 4602      4722  4      | IF .EVENT [EVENT$B_OPCODE] NEQU BPT_INSTRUCTION
: 4603      4723  4      | THEN
: 4604      4724  5      |     IF NOT WRITE_OPCODE (.EVENT_ADDRESS, BPT_INSTRUCTION)
: 4605      4725  4      |     THEN
: 4606      4726  5      |         BEGIN
: 4607      4727  5      |             SIGNAL (DBG$STEPINTO, 1, .DBG$RUNFRAME[DBG$L_USER_PC]);
: 4608      4728  5      |             DBG$RUNFRAME[DBG$V_TBIT] = TRUE;
: 4609      4729  5      |             EVENT [EVENT$V_STEP_BPT] = FALSE;
: 4610      4730  5      |             RETURN;
: 4611      4731  4      |             END;
: 4612      4732  4      |         END
: 4613      4733  3      |     ELSE
: 4614      4734  3      |         We're not stepping over, or we aren't going to BPT, so we'll
: 4615      4735  3      |         TBIT.
: 4616      4736  3      |         BEGIN
: 4617      4737  3      |             EVENT [EVENT$V_STEP_BPT] = FALSE;
: 4618      4738  4      |             DBG$RUNFRAME [DBG$V_TBIT] = TRUE;
: 4619      4739  4      |             END;
: 4620      4740  4      |         END;
: 4621      4741  3      |
: 4622      4742  2      |
: 4623      4743  2      |
: 4624      4744  2      |
: 4625      4745  2      | [EVENT$K_KIND_EXC]:
: 4626      4746  2      | TES;
: 4627      4747  2      |
: 4628      4748  1      | END;
```

			OFFC 00000	.ENTRY	DBG\$ACTIVATE_EVENT, Save R2,R3,R4,R5,R6,R7,-;	4253
					R8,R9,R10,R11	
		5B 00000000G	00 9E 00002	MOVAB	DBG\$RUNFRAME+64, R11	
		5E	14 C2 00009	SUBL2	#20, SP	
		56 04	AC D0 0000C	MOVL	EVENT, R6	4286
		59 14	A6 9E 00010	MOVAB	20(R6), R9	
		00 01	A9 8F 00014	CASEB	1(R9), #0, #2	
	02	01EE	0006 00019 1\$:	.WORD	2\$-1\$,-	
					28\$-1\$,-	
					41\$-1\$	
	04	00	02 A9 8F 0001F 2\$:	CASEB	2(R9), #0, #4	4294
000B	02B9	02B9	02B9 00024 3\$:	.WORD	41\$-3\$,-	

51	00000000G	52	01	D0	000E0	13\$:	MOVL	#1, AT_LIBSIGNAL	4436
		00	02	C1	000E3	14\$:	ADDL3	#2, DBG\$GL_LIB_SIGNAL_ADDR, R1	4446
		51	50	D1	000EB		CMPL	USERS_PC, R1	
51	00000000G	00	0D	13	000EE		BEQL	15\$	
		51	02	C1	000F0		ADDL3	#2, DBG\$GL_LIB_STOP_ADDR, R1	4447
			50	D1	000F8		CMPL	USERS_PC, R1	
		52	03	12	000FB		BNEQ	16\$	
60		06	01	D0	000FD	15\$:	MOVL	#1, AT_LIBSIGNAL	4449
			00	0C	00100	16\$:	PROBER	#0, #6, (USERS_PC)	4457
		17	43	13	00104		BEQL	19\$	
			60	91	00106		CMPB	(USERS_PC), #23	4460
			3E	12	00109		BNEQ	19\$	
	FF	8F	A0	91	0010B		CMPB	1(USERS_PC), #255	4461
			37	12	00110		BNEQ	19\$	
	04	AE	A0	D0	00112		MOVL	2(USERS_PC), ADDRESS	4469
		50	04	AE	C0		ADDL2	ADDRESS, R0	4470
04	BE	04	A0	9E	0011B		MOVAB	6(R0), ADDRESS	
			00	0C	00120		PROBER	#0, #4, @ADDRESS	4473
			05	13	00125		BEQL	17\$	
	04	AE	BE	D0	00127		MOVL	@ADDRESS, ADDRESS	4475
		50	00	9E	0012C	17\$:	MOVAB	LIB\$SIGNAL+2, R0	4477
		50	04	AE	D1		CMPL	ADDRESS, R0	
			0D	13	00137		BEQL	18\$	
		50	00	9E	00139		MOVAB	LIB\$STOP+2, R0	4478
		50	04	AE	D1		CMPL	ADDRESS, R0	
			03	12	00144		BNEQ	19\$	
		52	01	D0	00146	18\$:	MOVL	#1, AT_LIBSIGNAL	4480
		2D	52	E9	00149	19\$:	BLBC	AT_LIBSIGNAL, 21\$	4484
		50	AA	D0	0014C		MOVL	16(USERS_FP), RET_PC	4494
	19	A6	04	88	00150		BISB2	#4, 25(R6)	4495
	01	A9	01	90	00154		MOVB	#1, 1(R9)	4496
	02	A9	1B	A6	90		MOVB	27(R6), 2(R9)	4497
			04	12	0015D		BNEQ	20\$	4498
	02	A9	08	90	0015F		MOVB	#8, 2(R9)	4500
	2C	A6	50	D0	00163	20\$:	MOVL	RET_PC, 44(R6)	4501
	30	A6	60	90	00167		MOVB	(RET_PC), 48(R6)	4502
	08	AE	03	D0	0016B		MOVL	#3, OP	4503
			01	DD	0016F		PUSHL	#1	
			0C	AE	9F		PUSHAB	OP	
			50	DD	00174		PUSHL	RET_PC	
			FF	03	31		BRW	6\$	
			56	DD	00179	21\$:	PUSHL	R6	4514
	0000V	CF	01	FB	0017B		CALLS	#1, MATCH RIGHT CALL_FRAME	
		5A	24	A6	D0		MOVL	36(R6), USERS_FP	4515
	05	AA	80	8F	88		BISB2	#128, 5(USERS_FP)	4516
				04	00189		RET		4393
		58	2C	A6	D0	22\$:	MOVL	44(R6), EVENT_ADDRESS	4528
	30	A6	68	90	0018E		MOVB	(EVENT_ADDRESS), 48(R6)	4534
		6B	58	D1	00192		CMPL	EVENT_ADDRESS, DBG\$RUNFRAME+64	4542
			41	12	00195		BNEQ	25\$	
			10	DD	00197		PUSHL	#16	4553
	00000000G	00	01	FB	00199		CALLS	#1, DBG\$GET MEMORY	
		57	50	D0	001A0		MOVL	R0, IGNOR_ENTRY	
	00000000'	EF	67	0E	001A3		INSQUE	(IGNOR_ENTRY), EVENT\$CMD_QUEUE	4554
		50	A7	9E	001AA		MOVAB	20(IGNOR_ENTRY), R0	4567
		52	04	AC	D0		MOVL	EVENT, R2	4565
05		A2	05	E1	001B2		BBC	#5, 23(R2), 23\$	

		60		04	90	001B7	MOVB	#4, (R0)	4567
		03		03	11	001BA	BRB	24\$	
	01	60		05	90	001BC	23\$: MOVB	#5, (R0)	4569
	03	A0	0400	8F	B0	001BF	24\$: MOVW	#1024, 1(R0)	4571
		A0		04	88	001C5	BISB2	#4, 3(R0)	4573
				52	DD	001C9	PUSHL	R2	4574
				57	DD	001CB	PUSHL	IGNOR ENTRY	
75	0000V	CF		02	FB	001CD	CALLS	#2, INSQUE1	
	17	A2		05	E1	001D2	BBC	#5, 23(R2), 32\$	4580
				04	001D7	RET			4583
		50	04	AC	D0	001D8	25\$: MOVL	EVENT, R0	4594
		03	30	A0	91	001DC	CMPB	48(R0), #3	
				01	12	001E0	BNEQ	26\$	
				04	001E2	RET			
	0C	AE		03	D0	001E3	26\$: MOVL	#3, OP	4596
			10	01	DD	001E7	PUSHL	#1	
				AE	9F	001E9	PUSHAB	OP	
				58	DD	001EC	PUSHL	EVENT ADDRESS	
	00000000G	00		03	FB	001EE	CALLS	#3, DBG\$WRITE_MEM	
		01		50	E9	001F5	BLBC	R0, 27\$	
				04	001F8	RET			
			000284B4	8F	DD	001F9	27\$: PUSHL	#165044	4599
	00000000G	00		01	FB	001FF	CALLS	#1, LIB\$SIGNAL	
				04	00206	RET			4598
		51		6B	D0	00207	28\$: MOVL	DBG\$RUNFRAME+64, R1	4611
		50		61	9A	0020A	MOVZBL	(R1), OPCODE	
		52	18	A6	9E	0020D	MOVAB	24(R6), R2	4614
03		62		03	E0	00211	BBS	#3, (R2), 30\$	
				00BD	31	00215	29\$: BRW	39\$	
				0A	E0	00218	30\$: BBS	#10, (R2), 37\$	4615
7C		62		50	E1	0021C	BBC	OPCODE, DBG\$OPCODES_STEP_OVER, 29\$	
F1	00000000'	EF		0A	E0	00224	BBS	#10, (R2), 37\$	4621
70		62		50	D1	00228	CMPB	OPCODE, #250	4624
	000000FA	8F		09	13	0022F	BEQL	31\$	
	000000FB	8F		50	D1	00231	CMPB	OPCODE, #251	
				1E	12	00238	BNEQ	33\$	
	01	A2		04	88	0023A	31\$: BISB2	#4, 1(R2)	4636
			24	A6	D4	0023E	CLRL	36(R6)	4637
	03	A2	02	A9	90	00241	MOVB	2(R9), 3(R2)	4639
	01	A9	0400	8F	B0	00246	MOVW	#1024, 1(R9)	4638
	04	AB		10	88	0024C	32\$: BISB2	#16, DBG\$RUNFRAME+68	4641
	00000000G	00		04	88	00250	BISB2	#4, DBG\$GV_CONTROL+1	4642
				04	00257	RET			4626
				7E	7C	00258	33\$: CLRQ	-(SP)	4651
				51	DD	0025A	PUSHL	R1	4652
	00000000G	00		03	FB	0025C	CALLS	#3, DBG\$INS_DECODE	
		58		50	D0	00263	MOVL	R0, EVENT ADDRESS	
	40000000	8F		58	D1	00266	CMPB	EVENT_ADDRESS, #1073741824	4657
				04	1E	0026D	BGEQU	34\$	
				58	D5	0026F	TSTL	EVENT_ADDRESS	4658
				1B	12	00271	BNEQ	36\$	
				62	95	00273	34\$: TSTB	(R2)	4671
				04	19	00275	BLSS	35\$	
				58	D5	00277	TSTL	EVENT_ADDRESS	4672
				5E	12	00279	BNEQ	40\$	
				6B	DD	0027B	35\$: PUSHL	DBG\$RUNFRAME+64	4674
				01	DD	0027D	PUSHL	#1	

DBGEVENT
V04-000

M 15
16-Sep-1984 00:59:10
14-Sep-1984 12:16:53

VAX-11 Bliss-32 V4.0-742
[DEBUG.SRC]DBGEVENT.B32;1

Page 134
(14)

00000000G	00	00028063	8F	DD	0027F	PUSHL	#163939	:	
			03	FB	00285	CALLS	#3, LIB\$SIGNAL	:	
			4B	11	0028C	BRB	40\$:	4679
2C	A6		58	D0	0028E	36\$:	MOVL	EVENT ADDRESS, 44(R6)	4689
01	A2		04	88	00292	BISB2	#4, 1(R2)	:	4694
			04	11	00296	BRB	38\$:	4621
	58	2C	A6	D0	00298	37\$:	MOVL	44(R6), EVENT ADDRESS	4703
30	A6		68	90	0029C	38\$:	MOVB	(EVENT ADDRESS), 48(R6)	4715
	03	30	A6	91	002A0	CMPB	48(R6), #3	:	4722
			37	13	002A4	BEQL	41\$:	
10	AE		03	D0	002A6	MOVL	#3, OP	:	4724
			01	DD	002AA	PUSHL	#1	:	
		14	AE	9F	002AC	PUSHAB	OP	:	
00000000G	00		58	DD	002AF	PUSHL	EVENT ADDRESS	:	
	22		03	FB	002B1	CALLS	#3, DBG\$WRITE_MEM	:	
			50	E8	002B8	BLBS	R0, 41\$:	
			6B	DD	002BB	PUSHL	DBG\$RUNFRAME+64	:	4727
			01	DD	002BD	PUSHL	#1	:	
00000000G	00	00028063	8F	DD	002BF	PUSHL	#163939	:	
			03	FB	002C5	CALLS	#3, LIB\$SIGNAL	:	
04	AB		10	88	002CC	BISB2	#16, DBG\$RUNFRAME+68	:	4728
01	A2		04	8A	002D0	BICB2	#4, 1(R2)	:	4729
				04	002D4	RET		:	4726
	A2		04	8A	002D5	39\$:	BICB2	#4, 1(R2)	4739
04	AB		10	88	002D9	40\$:	BISB2	#16, DBG\$RUNFRAME+68	4740
			04	002DD	41\$:	RET		:	4748

; Routine Size: 734 bytes, Routine Base: DBG\$CODE + 1A4C


```
4630 4749 1 GLOBAL ROUTINE DBG$DEACTIVATE_EVENTS: NOVALUE =
4631 4750 1
4632 4751 1 FUNCTION
4633 4752 1     This routine deactivates all set events.
4634 4753 1
4635 4754 1 INPUTS
4636 4755 1     NONE
4637 4756 1
4638 4757 1 OUTPUTS
4639 4758 1     The events are deactivated, including the resetting of breakpoints, the
4640 4759 1     resetting of protections, etc.
4641 4760 1
4642 4761 1
4643 4762 2 BEGIN
4644 4763 2
4645 4764 2 LOCAL
4646 4765 2     EVENT_ENTRY : REF EVENT$EVENT_DESCRIPTOR, ! Event entry pointer
4647 4766 2     PAGE_ENTRY : REF EVENT$PAGE_DESCRIPTOR; ! Page entry pointer
4648 4767 2
4649 4768 2
4650 4769 2 ! As long as were not skipping ACCVIO's, for each entry in the page
4651 4770 2 ! queue, un-write protect the page described.
4652 4771 2
4653 4772 2 IF NOT .SKIP_ACCVIOS
4654 4773 2 THEN
4655 4774 3 BEGIN
4656 4775 3     PAGE_ENTRY = .EVENT$PAGE_QUEUE [L_QUEUE_FLINK];
4657 4776 3     WHILE .PAGE_ENTRY NEQA EVENT$PAGE_QUEUE DO
4658 4777 4         BEGIN
4659 4778 4             LOCAL
4660 4779 4                 ADDRESS : VECTOR [2]; ! Address range for protect
4661 4780 4
4662 4781 4                 ADDRESS [0] = .PAGE_ENTRY [EVENT$PAGE_ADDRESS];
4663 4782 4                 ADDRESS [1] = .PAGE_ENTRY [EVENT$PAGE_ADDRESS];
4664 4783 4                 IF NOT $SETPRT (INADR = ADDRESS,
4665 4784 4                     PROT = .PAGE_ENTRY [EVENT$PAGE_PROTECTION]
4666 4785 4                     )
4667 4786 4                     THEN
4668 4787 4                         SIGNAL (DBG$NOWPROT)
4669 4788 4                     ELSE
4670 4789 4                         PAGE_ENTRY = .PAGE_ENTRY [EVENT$PAGE_FLINK];
4671 4790 3                     END;
4672 4791 3     END;
4673 4792 2
4674 4793 2
4675 4794 2 ! Deactivate the all events not 'deleted'.
4676 4795 2
4677 4796 2 EVENT_ENTRY = .EVENT$CMD_QUEUE [L_QUEUE_FLINK];
4678 4797 2 WHILE .EVENT_ENTRY NEQA EVENT$CMD_QUEUE [A_QUEUE_ENTRY] DO
4679 4798 3 BEGIN
4680 4799 3     IF NOT .EVENT_ENTRY [EVENT$V_DELETED]
4681 4800 3     THEN
4682 4801 3         DBG$DEACTIVATE_EVENT (.EVENT_ENTRY);
4683 4802 3         EVENT_ENTRY = .EVENT_ENTRY [EVENT$CMD_FLINK];
4684 4803 2     END;
4685 4804 1 END;
```


			000C 00000	.ENTRY	DBG\$DEACTIVATE EVENTS, Save R2,R3	: 4749
53	00000000'	EF	9E 00002	MOVAB	EVENT\$PAGE_QUEUE, R3	: 4750
5E		08	C2 00009	SUBL2	#8, SP	: 4751
3D	00000000'	EF	E8 0000C	BLBS	SKIP ACCVIOS, 3\$: 4772
52		63	D0 00013	MOVL	EVENT\$PAGE_QUEUE, PAGE_ENTRY	: 4775
50		63	9E 00016	MOVAB	EVENT\$PAGE_QUEUE, R0	: 4776
50		52	D1 00019	CMPL	PAGE_ENTRY, R0	: 4777
		32	13 0001C	BEQL	3\$: 4778
04	6E	08	A2 D0 0001E	MOVL	8(PAGE_ENTRY), ADDRESS	: 4781
	AE	08	A2 D0 00022	MOVL	8(PAGE_ENTRY), ADDRESS+4	: 4782
	7E	0E	7E D4 00027	CLRL	-(SP)	: 4785
			A2 9A 00029	MOVZBL	14(PAGE_ENTRY), -(SP)	: 4786
			7E 7C 0002D	CLRQ	-(SP)	: 4787
		10	AE 9F 0002F	PUSHAB	ADDRESS	: 4788
00000000G	00	05	FB 00032	CALLS	#5, SYS\$SETPRT	: 4789
	0F	50	E8 00039	BLBS	R0, 2\$: 4790
00000000G	00	8F	DD 0003C	PUSHL	#165060	: 4787
		01	FB 00042	CALLS	#1, LIB\$SIGNAL	: 4788
		CB	11 00049	BRB	1\$: 4789
	52	62	D0 0004B	MOVL	(PAGE_ENTRY), PAGE_ENTRY	: 4789
		C6	11 0004E	BRB	1\$: 4776
	52	D0	A3 D0 00050	MOVL	EVENT\$CMD_QUEUE, EVENT_ENTRY	: 4796
	50	D0	A3 9E 00054	MOVAB	EVENT\$CMD_QUEUE, R0	: 4797
	50		52 D1 00058	CMPL	EVENT_ENTRY, R0	: 4798
		11	13 0005B	BEQL	6\$: 4799
		17	A2 95 0005D	TSTB	23(EVENT_ENTRY)	: 4799
		07	19 00060	BLSS	5\$: 4800
		52	DD 00062	PUSHL	EVENT_ENTRY	: 4801
0000V	CF	01	FB 00064	CALLS	#1, DBG\$DEACTIVATE EVENT	: 4802
	52	62	D0 00069	MOVL	(EVENT_ENTRY), EVENT_ENTRY	: 4802
		E6	11 0006C	BRB	4\$: 4797
		04	0006E	RET		: 4804

; Routine Size: 111 bytes, Routine Base: DBG\$CODE + 1D2A


```
4687 4805 1 GLOBAL ROUTINE DBGS$DEACTIVATE_EVENT(EVENT_ENTRY): NOVALUE =
4688 4806 1
4689 4807 1 FUNCTION
4690 4808 1     This routine deactivates an event.
4691 4809 1
4692 4810 1 INPUTS
4693 4811 1     EVENT_ENTRY :          Pointer to an event entry.
4694 4812 1
4695 4813 1 OUTPUTS
4696 4814 1     The event is deactivated, including the resetting of breakpoints, the
4697 4815 1     resetting of protections, etc.
4698 4816 1
4699 4817 1
4700 4818 2 BEGIN
4701 4819 2
4702 4820 2 MAP
4703 4821 2     EVENT_ENTRY :    REF EVENT$EVENT_DESCRIPTOR; ! Event entry pointer
4704 4822 2
4705 4823 2 LOCAL
4706 4824 2     EVENT_ADDRESS : REF VECTOR [, BYTE],      ! Event byte address
4707 4825 2     USERS_FP :      REF BLOCK [, BYTE],        ! User's current FP
4708 4826 2     STATUS;                          ! Status value
4709 4827 2
4710 4828 2
4711 4829 2     ! If this is an IGNOR entry, ignore it !
4712 4830 2
4713 4831 2 IF .EVENT_ENTRY [EVENT$B_CMD_TYPE] EQLU EVENT$K_TYPE_IGNORE
4714 4832 2 THEN
4715 4833 2     RETURN;
4716 4834 2
4717 4835 2
4718 4836 2     ! Case on the kind of event entry, to determine whether how to activate
4719 4837 2     the entry.
4720 4838 2
4721 4839 2 CASE .EVENT_ENTRY [EVENT$B_CMD_KIND]
4722 4840 2 FROM EVENT$R_KIND_ACC TO EVENT$K_KIND_EXC OF
4723 4841 2 SET
4724 4842 2     [EVENT$K_KIND_ACC]:
4725 4843 2
4726 4844 2
4727 4845 2     ! Case on the access sub-kind....
4728 4846 2
4729 4847 2 CASE .EVENT_ENTRY [EVENT$B_SUB_KIND]
4730 4848 2 FROM EVENT$R_ACC_READ TO EVENT$K_ACC_RTRN OF
4731 4849 2 SET
4732 4850 2
4733 4851 2
4734 4852 2     [EVENT$K_ACC_READ]:
4735 4853 2     ;
4736 4854 2
4737 4855 2     [EVENT$K_ACC_WRIT]:
4738 4856 2     ;
4739 4857 2
4740 4858 2
4741 4859 2     [EVENT$K_ACC_MDFY]:
4742 4860 2     ;
4743 4861 2
```


:	4744	4862	2
:	4745	4863	2
:	4746	4864	2
:	4747	4865	2
:	4748	4866	2
:	4749	4867	2
:	4750	4868	3
:	4751	4869	3
:	4752	4870	3
:	4753	4871	3
:	4754	4872	3
:	4755	4873	3
:	4756	4874	3
:	4757	4875	3
:	4758	4876	3
:	4759	4877	3
:	4760	4878	3
:	4761	4879	3
:	4762	4880	3
:	4763	4881	3
:	4764	4882	3
:	4765	4883	3
:	4766	4884	4
:	4767	4885	3
:	4768	4886	4
:	4769	4887	4
:	4770	4888	4
:	4771	4889	3
:	4772	4890	2
:	4773	4891	2
:	4774	4892	2
:	4775	4893	2
:	4776	4894	2
:	4777	4895	2
:	4778	4896	2
:	4779	4897	2
:	4780	4898	2
:	4781	4899	3
:	4782	4900	3
:	4783	4901	3
:	4784	4902	3
:	4785	4903	4
:	4786	4904	4
:	4787	4905	4
:	4788	4906	4
:	4789	4907	3
:	4790	4908	3
:	4791	4909	3
:	4792	4910	3
:	4793	4911	3
:	4794	4912	4
:	4795	4913	4
:	4796	4914	3
:	4797	4915	4
:	4798	4916	4
:	4799	4917	3
:	4800	4918	4

! Execution Access means that we've got to replace a BPT
instruction at the address indicated.

[EVENT\$K_ACC_EXEC]:
BEGIN

! Get the event's address.

EVENT_ADDRESS = .EVENT_ENTRY [EVENT\$L_ADDRESS];

! Finally, overwrite the BPT instruction with the original
opcode into that address. Note that we will NOT do this
if the original is a BPT.

IF .EVENT_ENTRY [EVENT\$B_OPCODE] NEQU BPT_INSTRUCTION
THEN
IF NOT WRITE_OPCODE (.EVENT_ADDRESS,
 .EVENT_ENTRY [EVENT\$B_OPCODE])

THEN
BEGIN
SIGNAL (DBG\$_NOWOPCO);
RETURN;
END;

END;

! Return Access means that we've got to replace a BPT
instruction at the address indicated - or, if this
is was a STEP command, turn of the dirty bit in the
user's PSW.

[EVENT\$K_ACC_RTRN]:
BEGIN

IF .EVENT_ENTRY [EVENT\$B_CMD_TYPE] EQLU
EVENT\$K_TYPE_SSINT

THEN
BEGIN
LOCAL BIT_15_CNT: REF VECTOR[.BYTE];
BIT_15_CNT = .EVENT_ENTRY [EVENT\$L_FP_RET_CNT];
IF .BIT_15_CNT[0] GEQ 1 THEN DBG\$GB_SET_WATCH_FLAG = TRUE;
END;

! Depending on whether this was a STEP command or not...

IF (.EVENT_ENTRY [EVENT\$B_CMD_TYPE] EQLU
EVENT\$K_TYPE_STEPS
) OR
(.EVENT_ENTRY [EVENT\$B_CMD_TYPE] EQLU
EVENT\$K_TYPE_SKIPS
) OR
(.EVENT_ENTRY [EVENT\$B_CMD_TYPE] EQLU


```

: 4801      4919  4      EVENT$K_TYPE_SSINT
: 4802      4920  4      )
: 4803      4921  3      THEN
: 4804      4922  4      BEGIN
: 4805      4923  4
: 4806      4924  4
: 4807      4925  4      ! Turn off bit 15 in the user's
: 4808      4926  4      ! runframe so we don't get another
: 4809      4927  4      ! ROPRAND fault on the return.
: 4810      4928  4
: 4811      4929  4      USERS_FP = .EVENT_ENTRY [EVENT$L_USERS_FP];
: 4812      4930  4      IF .USERS_FP NEQ 0
: 4813      4931  4      THEN
: 4814      4932  5      BEGIN
: 4815      4933  5      MATCH_RIGHT_CALL_FRAME(.EVENT_ENTRY);
: 4816      4934  5      USERS_FP = .EVENT_ENTRY [EVENT$L_USERS_FP];
: 4817      4935  5      (.USERS_FP [SF$W_SAVE_PSW])<15,1,0> = FALSE;
: 4818      4936  4      END;
: 4819      4937  4      END
: 4820      4938  3      ELSE
: 4821      4939  4      BEGIN
: 4822      4940  4
: 4823      4941  4      ! Get the event's address.
: 4824      4942  4      !
: 4825      4943  4      EVENT_ADDRESS = .EVENT_ENTRY [EVENT$L_ADDRESS];
: 4826      4944  4
: 4827      4945  4
: 4828      4946  4
: 4829      4947  4      ! Finally, overwrite the BPT instruction with the
: 4830      4948  4      ! original opcode into that address. Note that we
: 4831      4949  4      ! will NOT do this if the original is a BPT.
: 4832      4950  4
: 4833      4951  4      IF .EVENT_ENTRY [EVENT$B_OPCODE] NEQU BPT_INSTRUCTION
: 4834      4952  4      THEN
: 4835      4953  4      IF NOT WRITE_OPCODE (.EVENT_ADDRESS,
: 4836      4954  4      ! .EVENT_ENTRY [EVENT$B_OPCODE]
: 4837      4955  5      )
: 4838      4956  4      THEN
: 4839      4957  5      BEGIN
: 4840      4958  5      SIGNAL (DBG$NOWOPCO);
: 4841      4959  5      RETURN;
: 4842      4960  4      END;
: 4843      4961  3      END;
: 4844      4962  2      END;
: 4845      4963  2      TES;
: 4846      4964  2
: 4847      4965  2
: 4848      4966  2      [EVENT$K_KIND_INS]:
: 4849      4967  3      BEGIN
: 4850      4968  3
: 4851      4969  3
: 4852      4970  3      ! If this step was stepping over (had a BPT set), remove the
: 4853      4971  3      ! BPT instruction.
: 4854      4972  3      !
: 4855      4973  3      IF .EVENT_ENTRY [EVENT$V_STEP_OVER] AND
: 4856      4974  3      ! .EVENT_ENTRY [EVENT$V_STEP_BPT]
: 4857      4975  3      THEN
```



```
: 4858      4976  4      BEGIN
: 4859      4977  4
: 4860      4978  4
: 4861      4979  4      | Get the event's address.
: 4862      4980  4      EVENT_ADDRESS = .EVENT_ENTRY [EVENT$L_ADDRESS];
: 4863      4981  4
: 4864      4982  4
: 4865      4983  4
: 4866      4984  4      | Finally, overwrite the BPT instruction with the original
: 4867      4985  4      | opcode into that address. Note that we will NOT do this
: 4868      4986  4      | if the original is a BPT.
: 4869      4987  4
: 4870      4988  4      IF .EVENT_ENTRY [EVENT$B_OPCODE] NEQU BPT_INSTRUCTION
: 4871      4989  4      THEN
: 4872      P 4990  4          IF NOT WRITE_OPCODE (.EVENT_ADDRESS,
: 4873      P 4991  4              .EVENT_ENTRY [EVENT$B_OPCODE]
: 4874      4992  5              )
: 4875      4993  4          THEN
: 4876      4994  5              BEGIN
: 4877      4995  5              SIGNAL (DBG$_NOWOPCO);
: 4878      4996  5              RETURN;
: 4879      4997  4              END;
: 4880      4998  3      END;
: 4881      4999  2      END;
: 4882      5000  2
: 4883      5001  2      [EVENT$K_KIND_EXC]:
: 4884      5002  2      TES;
: 4885      5003  2
: 4886      5004  2      END;
: 4887      5005  1
```

				003C 00000	.ENTRY	DBG\$DEACTIVATE_EVENT, Save R2,R3,R4,R5	: 4805
		55	00000000G	00 9E 00002	MOVAB	DBG\$WRITE_MEM, R5	
		5E		0C C2 00009	SUBL2	#12, SP	
		52	04 AC D0 0000C	MOV L	EVENT_ENTRY, R2		: 4831
		05	14 A2 91 00010	CMPB	20(R2), #5		
			7E 13 00014	BEQL	10\$		
	02	00	15 A2 8F 00016	CASEB	21(R2), #0, #2		: 4839
	00BA	0087	0006 0001B 1\$:	.WORD	2\$-1\$,-		
					11\$-1\$,-		
					14\$-1\$		
	04	00	16 A2 8F 00021 2\$:	CASEB	22(R2), #0, #4		: 4847
000B	00AF	00AF	00AF 00026 3\$:	.WORD	14\$-3\$,-		
		0027	0002E		14\$-3\$,-		
					14\$-3\$,-		
					4\$-3\$,-		
					6\$-3\$		
		54	2C A2 D0 00030	RET			
		03	30 A2 91 00031 4\$:	MOV L	44(R2), EVENT_ADDRESS		: 4873
			79 13 00035	CMPB	48(R2), #3		: 4880
		6E	30 A2 9A 0003B	BEQL	12\$		
			01 DD 0003F	MOVZBL	48(R2), OP		: 4884
				PUSHL	#1		

		04	AE	9F	00041		PUSHAB	OP		
			54	DD	00044	5\$:	PUSHL	EVENT ADDRESS		
	65		03	FB	00046		CALLS	#3, DBG\$WRITE_MEM		
	7C		50	E9	00049		BLBC	R0, 13\$		
				04	0004C		RET			4887
			51	D4	0004D	6\$:	CLRL	R1		4900
	06	14	A2	91	0004F		CMPB	20(R2), #6		
			11	12	00053		BNEQ	7\$		
			51	D6	00055		INCL	R1		
	50	2C	A2	D0	00057		MOVL	44(R2), BIT_15_CNT		4905
			60	95	0005B		TSTB	(BIT_15_CNT)		4906
			07	13	0005D		BEQL	7\$		
00000000'	EF		01	90	0005F		MOVB	#1, DBG\$GB_SET_WATCH_FLAG		
	03	14	A2	91	00066	7\$:	CMPB	20(R2), #3		4912
			09	13	0006A		BEQL	8\$		
	04	14	A2	91	0006C		CMPB	20(R2), #4		4915
			03	13	00070		BEQL	8\$		
	17		51	E9	00072		BLBC	R1, 9\$		4918
	53	24	A2	D0	00075	8\$:	MOVL	36(R2), USERS_FP		4929
			5A	13	00079		BEQL	14\$		4930
			52	DD	0007B		PUSHL	R2		4933
0000V	CF		01	FB	0007D		CALLS	#1, MATCH_RIGHT_CALL_FRAME		
	53	24	A2	D0	00082		MOVL	36(R2), USERS_FP		4934
05	A3	80	8F	8A	00086		BICB2	#128, 5(USERS_FP)		4935
				04	0008B		RET			4912
	54	2C	A2	D0	0008C	9\$:	MOVL	44(R2), EVENT_ADDRESS		4944
	03	30	A2	91	00090		CMPB	48(R2), #3		4951
			3F	13	00094	10\$:	BEQL	14\$		
04	AE	30	A2	9A	00096		MOVZBL	48(R2), OP		4955
			01	DD	0009B		PUSHL	#1		
		08	AE	9F	0009D		PUSHAB	OP		
			A2	11	000A0		BRB	5\$		
2E	18	A2	03	E1	000A2	11\$:	BBC	#3, 24(R2), 14\$		4973
29	19	A2	02	E1	000A7		BBC	#2, 25(R2), 14\$		4974
		2C	A2	D0	000AC		MOVL	44(R2), EVENT_ADDRESS		4981
	54	30	A2	91	000B0		CMPB	48(R2), #3		4988
	03		1F	13	000B4	12\$:	BEQL	14\$		
	08	AE	30	A2	9A	000B6	MOVZBL	48(R2), OP		4992
			01	DD	000BB		PUSHL	#1		
		0C	AE	9F	000BD		PUSHAB	OP		
			54	DD	000C0		PUSHL	EVENT ADDRESS		
	65		03	FB	000C2		CALLS	#3, DBG\$WRITE_MEM		
	0D		50	E8	000C5		BLBS	R0, 14\$		
00000000G	00	000284BC	8F	DD	000C8	13\$:	PUSHL	#165052		4995
			01	FB	000CE		CALLS	#1, LIB\$SIGNAL		
			04	000D5	14\$:		RET			5005

; Routine Size: 214 bytes, Routine Base: DBG\$CODE + 1D99


```
: 4889      5006 1 GLOBAL ROUTINE DBG$EXCEPTION_HANDLER(SIGNAL_ARG_PTR, MECHAN_ARG_PTR) =
: 4890      5007 1
: 4891      5008 1 FUNCTIONAL DESCRIPTION:
: 4892      5009 1
: 4893      5010 1 Exception analyzer called by the primary vector exception handler,
: 4894      5011 1 which is a MARS routine found in DBGSTART.MAR. The MARS routine
: 4895      5012 1 immediately resignals if the user program was not running. Otherwise
: 4896      5013 1 it saves the registers of the user program and disables ASTs for the
: 4897      5014 1 time that DEBUG is running.
: 4898      5015 1
: 4899      5016 1 Then it calls this routine, where the exception is analyzed for the
: 4900      5017 1 type of exception. Breakpoint, trace, and ACCVIO traps are given
: 4901      5018 1 special handling, which usually ends with control being passed to the
: 4902      5019 1 user. If the breakpoint, trace, or ACCVIO trap was illegal, then the
: 4903      5020 1 exception is resignaled unless the user has asked for control on every
: 4904      5021 1 exception.
: 4905      5022 1
: 4906      5023 1 Some trace traps cause an interim halt that requires some action, but
: 4907      5024 1 doesn't pass control back to the user. After checking the validity of
: 4908      5025 1 these trace traps, the value SS$CONTINUE is returned.
: 4909      5026 1
: 4910      5027 1 After the exception is analyzed and it is determined that immediate
: 4911      5028 1 resignaling or continuing is not desired, the user_proc routine is
: 4912      5029 1 called. This routine accepts user commands either from prespecified
: 4913      5030 1 action commands from breakpoints, or interactively from the terminal.
: 4914      5031 1 Eventually, a command is given that either causes the user program to
: 4915      5032 1 continue or DEBUG to exit. If the user program is to continue, the
: 4916      5033 1 value returned from user_proc is SS$CONTINUE, and that value is
: 4917      5034 1 passed back to the MARS handler.
: 4918      5035 1
: 4919      5036 1 If an exception occurs during DEBUG processing, the exception handler
: 4920      5037 1 is FINAL_HANDL, not this routine.
: 4921      5038 1
: 4922      5039 1 FORMAL PARAMETERS:
: 4923      5040 1
: 4924      5041 1 SIGNAL_ARG_PTR - address of block that contains at least four
: 4925      5042 1 longwords. The pertinent words are the exception
: 4926      5043 1 name, the PC at the time of the exception, and the PSL
: 4927      5044 1 at the time of the exception. The name is always the
: 4928      5045 1 second longword, the PC and the PSL the next to last
: 4929      5046 1 and last respectively.
: 4930      5047 1
: 4931      5048 1 MECHAN_ARG_PTR - address of block that contains five longwords. The
: 4932      5049 1 pertinent words are the saved R0 and R1. They are in
: 4933      5050 1 the fourth and fifth longwords respectively. Neither
: 4934      5051 1 is used at this time.
: 4935      5052 1
: 4936      5053 1 IMPLICIT INPUTS:
: 4937      5054 1
: 4938      5055 1 Some variable number of additional arguments may exist between the
: 4939      5056 1 exception name and the PC. Flags indicating the validity of IBITs and
: 4940      5057 1 breakpoints are referenced. The flag DBG$GB_RESIGNAL causes illegal
: 4941      5058 1 exceptions to be resignaled if the flag is set to true.
: 4942      5059 1
: 4943      5060 1
: 4944      5061 1
: 4945      5062 1
```



```
: 4946      5063  1  | IMPLICIT OUTPUTS:
: 4947      5064  1  |
: 4948      5065  1  |     The TBIT in the RUNFRAME PSL may be changed.
: 4949      5066  1  |
: 4950      5067  1  |
: 4951      5068  1  | ROUTINE VALUE:
: 4952      5069  1  |
: 4953      5070  1  |     $$$_RESIGNAL or $$$_CONTINUE for resignaling and continuing
: 4954      5071  1  |     respectively.
: 4955      5072  1  |
: 4956      5073  1  |
: 4957      5074  1  | SIDE EFFECTS:
: 4958      5075  1  |
: 4959      5076  1  |     Any number of things.
: 4960      5077  1  |
: 4961      5078  2  | BEGIN
: 4962      5079  2  | MAP
: 4963      5080  2  |     MECHAN_ARG_PTR : REF BLOCK[,BYTE],
: 4964      5081  2  |     SIGNAL_ARG_PTR : REF BLOCK[,BYTE];
: 4965      5082  2  | BUILTIN PROBER;
: 4966      5083  2  | LABEL scan;
: 4967      5084  2  | LOCAL
: 4968      5085  2  |     EVENT :          REF EVENT$EVENT_DESCRIPTOR, ! Event entry pointer
: 4969      5086  2  |     LAST1 :          REF EVENT$EVENT_DESCRIPTOR, ! Event entry pointer
: 4970      5087  2  |     SKIPS_ENTRY :    REF EVENT$EVENT_DESCRIPTOR, ! Skip entry pointer
: 4971      5088  2  |     ACTIVE_EVENT,    ! Event match flag
: 4972      5089  2  |     EXCEPTION_TYPE,
: 4973      5090  2  |     USERS_PC :       REF BLOCK [, BYTE],
: 4974      5091  2  |     USERS_FP :       REF BLOCK [, BYTE],
: 4975      5092  2  |     OPCODE,
: 4976      5093  2  |     ORIG_TBIT,       ! Original TBIT value
: 4977      5094  2  |     STATUS;
: 4978      5095  2  |
: 4979      5096  2  |
: 4980      5097  2  | ! Save the original value of the TBIT.
: 4981      5098  2  |
: 4982      5099  2  | ORIG_TBIT = .DBG$RUNFRAME[DBG$V_TBIT];
: 4983      5100  2  |
: 4984      5101  2  |
: 4985      5102  2  | ! Save the exception type (name);
: 4986      5103  2  |
: 4987      5104  2  | EXCEPTION_TYPE = .SIGNAL_ARG_PTR [CHF$SIG_NAME];
: 4988      5105  2  |
: 4989      5106  2  |
: 4990      5107  2  | ! Save the User's PC.
: 4991      5108  2  |
: 4992      5109  2  | USERS_PC = .DBG$RUNFRAME [DBG$L_USER_PC];
: 4993      5110  2  | IF NOT PROBER(%REF(0),%REF(1),.USERS_PC) THEN OPCODE = 0
: 4994      5111  4  | ELSE IF (((OPCODE = .(.USERS_PC)<0,8,0>) GEQU %X'FD')
: 4995      5112  3  |         AND PROBER(%REF(0),%REF(2),.USERS_PC))
: 4996      5113  2  | THEN OPCODE = %X'100' + .(.USERS_PC)<8,8,0>;
: 4997      5114  2  |
: 4998      5115  2  |
: 4999      5116  2  | ! If we got here via a TBIT, see if we're at one of the special DEBUG
: 5000      5117  2  | ! locations. If so, just return $$$_CONTINUE.
: 5001      5118  2  |
: 5002      5119  2  | IF .EXCEPTION_TYPE EQLU $$$_TBIT
```



```

: 5003      5120 2
: 5004      5121 2
: 5005      5122 2
: 5006      5123 2
: 5007      5124 2
: 5008      5125 2
: 5009      5126 2
: 5010      5127 2
: 5011      5128 4
: 5012      5129 4
: 5013      5130 4
: 5014      5131 4
: 5015      5132 2
: 5016      5133 2
: 5017      5134 2
: 5018      5135 2
: 5019      5136 2
: 5020      5137 2
: 5021      5138 2
: 5022      5139 2
: 5023      5140 2
: 5024      5141 2
: 5025      5142 2
: 5026      5143 2
: 5027      5144 2
: 5028      5145 2
: 5029      5146 2
: 5030      5147 2
: 5031      5148 2
: 5032      5149 2
: 5033      5150 2
: 5034      5151 2
: 5035      5152 2
: 5036      5153 2
: 5037      5154 2
: 5038      5155 2
: 5039      5156 2
: 5040      5157 2
: 5041      5158 2
: 5042      5159 2
: 5043      5160 2
: 5044      5161 2
: 5045      5162 2
: 5046      5163 2
: 5047      5164 2
: 5048      5165 2
: 5049      5166 2
: 5050      5167 2
: 5051      5168 2
: 5052      5169 2
: 5053      5170 2
: 5054      5171 2
: 5055      5172 2
: 5056      5173 2
: 5057      5174 2
: 5058      5175 2
: 5059      5176 2

THEN
BEGIN
  IF .USERS_PC EQLA DBG$PSEUDO_EXIT OR
    .USERS_PC EQLA DBG$TERM_HANDLER OR
    .USERS_PC EQLA DBG$USER_EXIT OR
    .USERS_PC EQLA PRIM_HANDLER_2 OR
    .USERS_PC EQLA DBG$PSEUDO_SSI
  THEN
    BEGIN
      DBG$RUNFRAME [DBG$V_TBIT] = FALSE;
      DBG$GL_OUTPRAB [RAB$V_CCO] = TRUE;
      RETURN SSS_CONTINUE;
    END;
  END;

  ! If we got here via an Opcode Request....
  IF .EXCEPTION_TYPE EQLU SSS_DBGOPCREQ
  THEN
    BEGIN

      ! This signal is generated by the library routine LIB$GET_OPCODE.
      ! It is used to inquire what the real instruction was before the
      ! debugger replaced the instruction with a BPT.

      ! (The emulator needs this capability).

      ! Note that the opcode involved was restored when we deactivated
      ! the events. This will have to be redone when we stop
      ! automatically deactivating....

      DBG$DEACTIVATE_EVENTS();
      (.SIGNAL_ARG_PTR [12,0,32,0]) <0, 8, 0> = (.SIGNAL_ARG_PTR [8,0,32,0]) <0, 8, 0>;
      DBG$ACTIVATE_EVENTS();
      RETURN SSS_CONTINUE;
    END;

    *****SSI
    ! If we got here via System service interception...
    ! Build a SSI entry on the Event List, if this system service calls from
    ! user program, and nobody else has turned on bit 15 in saved PSW in
    ! System Service FP. (This means that TDBG has seen this already, bit
    ! 15 is set, now is SDBG's turn to examine this signal). Record the PC
    ! to report watch At, unprotected the pages...

    IF .EXCEPTION_TYPE EQLU DBG$_SS_INT
    THEN
      BEGIN
        LOCAL
          ADDRESS : VECTOR [2],
          BIT_15_CNT : REF VECTOR[.BYTE],
          BIT_15_CNT_SSV : REF VECTOR[.BYTE],
          ! Address range for protect
          ! Has bit 15 turned on?
          ! Are we called from System
          ! Service's system service

```



```
5060      5177      3
5061      5178
5062      5179      3
5063      5180      3
5064      5181      3
5065      5182      3
5066      5183      3
5067      5184      3
5068      5185      3
5069      5186      3
5070      5187      3
5071      5188      3
5072      5189      3
5073      5190      3
5074      5191      3
5075      5192      3
5076      5193      3
5077      5194      3
5078      5195      3
5079      5196      3
5080      5197      3
5081      5198      3
5082      5199      3
5083      5200      3
5084      5201      3
5085      5202      3
5086      5203      3
5087      5204      3
5088      5205      3
5089      5206      3
5090      5207      3
5091      5208      3
5092      5209      3
5093      5210      3
5094      5211      3
5095      5212      3
5096      5213      3
5097      5214      3
5098      5215      3
5099      5216      3
5100      5217      3
5101      5218      3
5102      5219      3
5103      5220      3
5104      5221      3
5105      5222      3
5106      5223      3
5107      5224      3
5108      5225      3
5109      5226      4
5110      5227      4
5111      5228      4
5112      5229      5
5113      5230      5
5114      5231      5
5115      P 5232      5
5116      P 5233      5

! call?
PAGE_ENTRY: REF EVENT$PAGE_DESCRIPTOR, ! Page entry pointer
SIGARG: REF VECTOR[ LONG], ! Info. comes in with DBG$SS_INT
SSINT_ENTRY: REF EVENT$EVENT_DESCRIPTOR, ! SSI event entry
USERS_FP: REF BLOCK[ BYTE]; ! System service call frame

! Get arg. list.
SIGARG = SIGNAL_ARG_PTR[CHF$SIG_ARG1];

! Caused from System service's system call, simply ignore it.
! note: this variable is a communication variable, it is seen
! by DBG, TDBG and SDBG.
BIT_15_CNT_SSV = .SIGARG[5];
IF .BIT_15_CNT_SSV[0] GTR 1 THEN RETURN SS$_CONTINUE;

! This type of entry is already built for somebody else (caused from
! the same source, went through different level). Ignore it.
! note: this variable is a communication variable, it is seen
! by DBG, TDBG and SDBG.
BIT_15_CNT = .SIGARG[4];
IF .BIT_15_CNT[0] GEQ 1 THEN RETURN SS$_CONTINUE;

! We'll take it. Mark the cycle begins. Build event entry, place
! it at the head of the event list.
DBG$GB SET_WATCH_FLAG = TRUE;
USERS_FP = .SIGARG[3];
EVENT = .EVENT$CMD_QUEUE [L_QUEUE_FLINK];
SSINT_ENTRY = DBG$GET_MEMORY (EVENT$K_EVENT_DESCRIPTOR_SIZE);
INSQUE (.SSINT_ENTRY, .EVENT[EVENT$K_CMD_BLINK]);
SSINT_ENTRY [EVENT$B_CMD_TYPE] = EVENT$K_TYPE_SSINT;
SSINT_ENTRY [EVENT$B_CMD_KIND] = EVENT$K_KIND_ACC;
SSINT_ENTRY [EVENT$B_SUB_KIND] = EVENT$K_ACC_RTRN;
SSINT_ENTRY [EVENT$V_ONCE_ONLY] = TRUE;
SSINT_ENTRY [EVENT$K_EXC_FLINK] = .SSINT_ENTRY;
SSINT_ENTRY [EVENT$K_EXC_BLINK] = .SSINT_ENTRY;
SSINT_ENTRY [EVENT$K_USERS_FP] = .USERS_FP;
(USERS_FP [SFSW_SAVE_PSW]) < 15, 1, 0 > = TRUE;
SSINT_ENTRY [EVENT$K_SSV_COUNT] = .BIT_15_CNT_SSV;
SSINT_ENTRY [EVENT$K_FP_RET_CNT] = .BIT_15_CNT;
SSINT_ENTRY [EVENT$K_USERS_PC] = .USERS_FP[SFSL_SAVE_PC];
WATCH_PC = .USERS_FP[SFSL_SAVE_PC];
BIT_15_CNT[0] = .BIT_15_CNT[0] + 1;
IF .BIT_15_CNT[0] EQ 1
THEN
  BEGIN
    PAGE_ENTRY = .EVENT$PAGE_QUEUE [L_QUEUE_FLINK];
    WHILE .PAGE_ENTRY NEQA EVENT$PAGE_QUEUE DO
      BEGIN
        ADDRESS [0] = .PAGE_ENTRY [EVENT$K_PAGE_ADDRESS];
        ADDRESS [1] = .PAGE_ENTRY [EVENT$K_PAGE_ADDRESS];
        IF NOT $SETPRT (INADR = ADDRESS,
          PROT = .PAGE_ENTRY [EVENT$B_PAGE_PROTECTION])
```



```
5117      5234      6
5118      5235
5119      5236      THEN
5120      5237      SIGNAL (DBG$_NOWPROT)
5121      5238      ELSE
5122      5239      PAGE_ENTRY = .PAGE_ENTRY [EVENT$_PAGE_FLINK];
5123      5240      END;
5124      5241      END;
5125      5242      RETURN $$$_CONTINUE;
5126      5243      END;
5127      5244
5128      5245      ! If this is an exception break or trace, we'll flag it later.
5129      5246      !
5130      5247      DBG$_EXC_BRE_FLAG = FALSE;
5131      5248
5132      5249      ! Clear the event-match flag, to indicate that we have as yet found no
5133      5250      ! matching event entry for this exception.
5134      5251      !
5135      5252      ACTIVE_EVENT = FALSE;
5136      5253
5137      5254      ! Set the type-source flag, so that we will display the source line at
5138      5255      ! the first opportunity.
5139      5256      !
5140      5257      TYPE_SOURCE = TRUE;
5141      5258
5142      5259      ! Clear the STOP_USER flag, set when a break is valid, or step is over.
5143      5260      ! This flag is usually set in PROCESS_EVENT when a break breaks, or a
5144      5261      ! step stops....
5145      5262      !
5146      5263      EVENT$_STOP_USER = FALSE;
5147      5264
5148      5265      ! Set the flag allowing user commands, i.e. no GO, STEP, EXIT, or
5149      5266      ! CONTINUE commands have been executed yet. We have to do it here
5150      5267      ! in case there are no DO's evaluated, since we test it later....
5151      5268      !
5152      5269      DBG$_TAKE_CMD = TRUE;
5153      5270
5154      5271      ! Deactivate all events. This will most probably be changed to take
5155      5272      ! advantage of not HAVING to turn them all off, but later, LATER !
5156      5273      !
5157      5274      DBG$_DEACTIVATE_EVENTS ();
5158      5275      IF .EXCEPTION_TYPE EQLU $$$_DEBUG
5159      5276      THEN
5160      5277      BEGIN
5161      5278
5162      5279
5163      5280
5164      5281
5165      5282
5166      5283
5167      5284
5168      5285
5169      5286
5170      5287
5171      5288
5172      5289
5173      5290
5173      5290      ! This is the phony way into the user control. This exception name
5173      5290      ! is passed by the DEBUG startup routine, and by the final handler
5173      5290      ! when an exception generated in a user program has not been dealt
5173      5290      ! with by any user handlers (and so has been reported by the DEBUG
5173      5290      ! final handler), when the user is passed control in order to fix
```



```

5174
5175
5176
5177
5178
5179
5180
5181
5182
5183
5184
5185
5186
5187
5188
5189
5190
5191
5192
5193
5194
5195
5196
5197
5198
5199
5200
5201
5202
5203
5204
5205
5206
5207
5208
5209
5210
5211
5212
5213
5214
5215
5216
5217
5218
5219
5220
5221
5222
5223
5224
5225
5226
5227
5228
5229
5230

```

```

5291 3      ! up the condition that caused the problem.
5292 3
5293 3      ! We'll act as though a BREAKpoint was encountered.
5294 3
5295 3      EVENT$B_STOP_USER = TRUE;
5296 3      ACTIVE_EVENT = TRUE;
5297 3      END;
5298 3
5299 3
5300 2      ! If the user program has completed, cancel any pending STEP events.
5301 2
5302 2      IF .DBG$RUNFRAME[DBG$SL_USER_PC] EQL 0
5303 2      THEN
5304 2          BEGIN
5305 2              EVENT = EVENT$CMD_QUEUE[A_QUEUE_ENTRY];
5306 2              WHILE TRUE DO
5307 2                  BEGIN
5308 2                      EVENT = .EVENT[EVENT$SL_CMD_FLINK];
5309 2                      IF .EVENT EQLA EVENT$CMD_QUEUE[A_QUEUE_ENTRY] THEN EXITLOOP;
5310 2                      IF (.EVENT[EVENT$B_CMD_TYPE] EQL EVENT$K_TYPE_STEPS)
5311 2                      THEN
5312 2                          EVENT[EVENT$V_DELETED] = TRUE;
5313 2
5314 2                      END;
5315 2
5316 2              END
5317 2
5318 2      ELSE
5319 2          BEGIN
5320 2
5321 2              ! Based on exception name, handle the exception....
5322 2
5323 2              SELECTONE .EXCEPTION TYPE OF
5324 2              SET
5325 2              [SS$ _ROPRAND, SS$ _DEBUG, SS$ _BREAK, SS$ _TBIT]:
5326 2              IF (.EXCEPTION_TYPE NEQ SS$ _DEBUG) OR
5327 2              .DBG$GV_CONTROL[DBG$V_CONTROL_STOP]
5328 2              THEN
5329 2                  BEGIN
5330 2
5331 2                      ! A BPT or TBIT gets us here. These are two of the
5332 2                      ! exceptions DEBUG uses to make events happen, and so
5333 2                      ! are treated specially here.
5334 2
5335 2                      ! Point to the current last entry on the queue. We'll only
5336 2                      ! examine entries until :
5337 2                      ! a) we're at the actual end of the queue, or
5338 2                      ! b) we've looked at the last entry we knew about when we started.
5339 2                      ! This is because processing a DO might add entries that we
5340 2                      ! then want to ignore....
5341 2
5342 2                      LAST1 = .EVENT$CMD_QUEUE [L_QUEUE_BLINK];

```



```
: 5231 5343 4
: 5232 5349 4
: 5233 5350 4
: 5234 5351 4
: 5235 5352 4
: 5236 5353 5
: 5237 5354 5
: 5238 5355 5
: 5239 5356 5
: 5240 5357 5
: 5241 5358 5
: 5242 5359 5
: 5243 5360 6
: 5244 5361 6
: 5245 5362 6
: 5246 5363 6
: 5247 5364 6
: 5248 5365 6
: 5249 5366 6
: 5250 5367 6
: 5251 5368 6
: 5252 5369 6
: 5253 5370 6
: 5254 5371 6
: 5255 5372 6
: 5256 5373 6
: 5257 5374 6
: 5258 5375 7
: 5259 5376 7
: 5260 5377 7
: 5261 5378 7
: 5262 5379 7
: 5263 5380 7
: 5264 5381 7
: 5265 5382 7
: 5266 5383 7
: 5267 5384 7
: 5268 5385 7
: 5269 5386 6
: 5270 5387 6
: 5271 5388 6
: 5272 5389 6
: 5273 5390 6
: 5274 5391 7
: 5275 5392 7
: 5276 5393 7
: 5277 5394 7
: 5278 5395 7
: 5279 5396 7
: 5280 5397 7
: 5281 5398 7
: 5282 5399 7
: 5283 5400 7
: 5284 5401 7
: 5285 5402 7
: 5286 5403 7
: 5287 5404 7
```

```
: For each event entry in the queue, check to see if it is
: active....
```

```
EVENT = .EVENT$CMD QUEUE [L_QUEUE-FLINK];
WHILE .EVENT NEQA EVENT$CMD-QUEUE[A_QUEUE-ENTRY] DO
  BEGIN
```

```
: Make sure that the entry isn't deleted but not removed....
```

```
IF NOT .EVENT [EVENT$V-DELETED]
THEN
  BEGIN
```

```
: If this entry is to be IGNOREd....
```

```
IF .EVENT [EVENT$B-CMD-TYPE] EQLU EVENT$K-TYPE-IGNOR
THEN
```

```
: IGNOREd entries are always deleted. If this wasn't
: a TBIT exception, the entry will be re-entered when
: the associated parent-event is re-activated ('cause
: all IGNOREd entries are used to TBIT over some
: instruction).
```

```
BEGIN
EVENT [EVENT$V-DELETED] = TRUE;
```

```
: Ignored entries are active if this is a TBIT.
```

```
IF .EXCEPTION-TYPE EQLU $$$-TBIT
THEN
  ACTIVE_EVENT = TRUE;
END
```

```
ELSE
```

```
: This is a 'real' event, not an IGNOREd one.
```

```
BEGIN
```

```
: Case on the kind of event entry, to determine whether
: the exception is applicable, and if the entry is
: active.
```

```
CASE .EVENT [EVENT$B-CMD-KIND]
FROM EVENT$K-KIND-ACC TO EVENT$K-KIND-EXC OF
  SET
  [EVENT$K-KIND-ACC]:
```

```
: Case on the access sub-kind....
```


DBGEVENT
V04-000

C 1
16-Sep-1984 00:59:10 VAX-11 Bliss-32 V4.0-742
14-Sep-1984 12:16:53 [DEBUG.SRC]DBGEVENT.B32;1

Page 149
(17)

```
: 5288      5405      7
: 5289      5406      7
: 5290      5407      7
: 5291      5408      7
: 5292      5409      7
: 5293      5410      7
: 5294      5411      7
: 5295      5412      7
: 5296      5413      7
: 5297      5414      7
: 5298      5415      7
: 5299      5416      7
: 5300      5417      7
: 5301      5418      7
: 5302      5419      7
: 5303      5420      7
: 5304      5421      7
: 5305      5422      7
: 5306      5423      7
: 5307      5424      7
: 5308      5425      7
: 5309      5426      7
: 5310      5427      7
: 5311      5428      7
: 5312      5429      7
: 5313      5430      7
: 5314      5431      7
: 5315      5432      7
: 5316      5433      7
: 5317      5434      8
: 5318      5435      8
: 5319      5436      8
: 5320      5437      8
: 5321      5438      8
: 5322      5439      8
: 5323      5440      8
: 5324      5441      8
: 5325      5442      8
: 5326      5443      8
: 5327      5444      8
: 5328      5445      8
: 5329      5446      8
: 5330      5447      8
: 5331      5448      8
: 5332      5449      8
: 5333      5450      8
: 5334      5451      8
: 5335      5452      8
: 5336      5453      8
: 5337      5454      8
: 5338      5455      8
: 5339      5456      8
: 5340      5457      8
: 5341      5458      8
: 5342      5459      8
: 5343      5460      8
: 5344      5461      8
```

```
!
CASE .EVENT [EVENT$B.SUB_KIND]
FROM EVENT$K_ACC_READ TO EVENT$K_ACC_RTRN OF
SET
```

```
[EVENT$K_ACC_READ]:
;
```

```
[EVENT$K_ACC_WRIT]:
;
```

```
[EVENT$K_ACC_MDFY]:
```

```
! If we're skipping ACCVIOS (meaning
! we had one, and it's page address
! was one we're watching for....
! ****SSI
! If we're returned from system
! service, or during system service
! cycle...
```

```
IF .SKIP_ACCVIOS OR
   .SKIP_WATCHES OR
   .DBG$GB_SET_WATCH_FLAG
THEN
  BEGIN
  LOCAL
  LENGTH;
```

```
! We've got an active event.
ACTIVE_EVENT = TRUE;
```

```
! Get the byte length of the
! entry's value. Note that we
! actually get its bit length and
! round up. This should be okay
! 'cause the values saved in a
! value descriptor are padded out
! to fill bytes.
```

```
LENGTH =
  DBG$DATA_LENGTH
  (EVENT [EVENT$A.VMSDESC]);
LENGTH = (.LENGTH + 7) * -3;
GET WATCH_POINT_VALUE(.EVENT);
OLDVALUE =
  .EVENT [EVENT$L.VALDESC];
```

```
! Now, compare the two values, and
```


DBGEVENT
V04-000

D 1
16-Sep-1984 00:59:10 VAX-11 Bliss-32 V4.0-742
14-Sep-1984 12:16:53 [DEBUG.SRC]DBGEVENT.B32;1

Page 150
(17)

```
.. 5345 5462 8
.. 5346 5463 8
.. 5347 5464 8
.. 5348 5465 8
.. 5349 5466 8
.. 5350 5467 8
.. 5351 5468 8
.. 5352 5469 8
.. 5353 5470 8
.. 5354 5471 8
.. 5355 5472 8
.. 5356 5473 8
.. 5357 5474 8
.. 5358 5475 9
.. 5359 5476 9
.. 5360 5477 9
.. 5361 5478 9
.. 5362 5479 9
.. 5363 5480 9
.. 5364 5481 9
.. 5365 5482 9
.. 5366 5483 9
.. 5367 5484 9
.. 5368 5485 9
.. 5369 5486 9
.. 5370 5487 9
.. 5371 5488 9
.. 5372 5489 9
.. 5373 5490 9
.. 5374 5491 9
.. 5375 5492 9
.. 5376 5493 9
.. 5377 5494 9
.. 5378 5495 9
.. 5379 5496 9
.. 5380 5497 9
.. 5381 5498 9
.. 5382 5499 9
.. 5383 5500 9
.. 5384 5501 9
.. 5385 5502 9
.. 5386 5503 9
.. 5387 5504 9
.. 5388 5505 9
.. 5389 5506 9
.. 5390 5507 9
.. 5391 5508 9
.. 5392 5509 8
.. 5393 5510 7
.. 5394 5511 7
.. 5395 5512 7
.. 5396 5513 7
.. 5397 5514 7
.. 5398 5515 7
.. 5399 5516 7
.. 5400 5517 8
.. 5401 5518 8
```

```
! process the event is they're the
! same.
```

```
IF CH$EQL
  (.LENGTH,
   OLDVALUE [DBG$A_VALUE_ADDRESS],
   .LENGTH,
   NEWVALUE [DBG$A_VALUE_ADDRESS],
   0
  )
```

```
THEN
  DBG$NFREE_DESC(.NEWVALUE)
```

```
ELSE
  BEGIN
```

```
! Process the entry.
```

```
IF .SKIP_WATCHES OR
   .DBG$GB_SET_WATCH_FLAG
THEN
  ACCVIO_PC = .WATCH_PC;
  STATUS = DBG$PROCESS_EVENT
           (.EVENT,
            .SIGNAL_ARG_PTR
           );
```

```
! Update the value descriptor in the event entry to refl
! new value.
! Then, since we are finished with the value descriptor
! free up the memory.
```

```
CH$MOVE
  (.LENGTH,
   NEWVALUE [DBG$A_VALUE_ADDRESS],
   OLDVALUE [DBG$A_VALUE_ADDRESS]
  );
```

```
DBG$NFREE_DESC(.NEWVALUE);
```

```
! If we have been pre-
! empted by a GO, STEP,
! CONTINUE, or EXIT, we will
! stop processing.
```

```
IF .STATUS
THEN
  EXITLOOP;
```

```
END;
END;
```

```
! Process an /EXECUTION (BREAK/TRACE)
! entry.
```

```
[EVENT$K_ACC_EXEC]:
  BEGIN
```


5402	5519	8
5403	5520	8
5404	5521	8
5405	5522	8
5406	5523	8
5407	5524	8
5408	5525	8
5409	5526	9
5410	5527	9
5411	5528	9
5412	5529	9
5413	5530	9
5414	5531	9
5415	5532	9
5416	5533	9
5417	5534	9
5418	5535	9
5419	5536	9
5420	5537	9
5421	5538	9
5422	5539	9
5423	5540	9
5424	5541	9
5425	5542	9
5426	5543	8
5427	5544	8
5428	5545	7
5429	5546	7
5430	5547	7
5431	5548	7
5432	5549	7
5433	5550	7
5434	5551	7
5435	5552	8
5436	5553	8
5437	5554	8
5438	5555	8
5439	5556	8
5440	5557	8
5441	5558	8
5442	5559	8
5443	5560	8
5444	5561	8
5445	5562	8
5446	5563	8
5447	5564	8
5448	5565	9
5449	5566	9
5450	5567	9
5451	5568	9
5452	5569	9
5453	5570	9
5454	5571	9
5455	5572	9
5456	5573	9
5457	5574	9
5458	5575	9

```
! Match the address of the eventpoint
! with the current PC for an active
! event...
IF .EVENT [EVENT$$_ADDRESS] EQ LA .USERS_PC
THEN
    BEGIN

        ! The event is active, so process
        ! the entry. If on return we have
        ! been pre-empted by a GO, STEP,
        ! CONTINUE, or EXIT, we will stop
        ! processing.
        ACTIVE_EVENT = TRUE;
        IF DBG$PROCESS_EVENT
            (.EVENT,
             .SIGNAL_ARG_PTR
            )
        THEN
            EXITLOOP;

        END;

    END;

! Process a /RETURN (BREAK/TRACE/STEPS/
! SKIPS) entry.
[EVENT$$_ACC_RTRN]:
    BEGIN

        ! Based on the event command type....
        !
        ! SELECT ONE .EVENT [EVENT$$_CMD_TYPE] OF
        ! SET

        ! [BREAK:TRACE]/RETURN
        !
        ! [EVENT$$_TYPE_BREAK,
        !  EVENT$$_TYPE_TRACE]:
        ! BEGIN

        ! Match the address of the
        ! eventpoint with the current
        ! PC for an active event...
        ! Also make sure the exception is not ROPRAND
        ! (this is in case the user sets a
        ! return break at the RET).
        IF (.EVENT [EVENT$$_ADDRESS] EQ LA .USERS_PC) AND
```


DBGEVENT
V04-000

F 1
16-Sep-1984 00:59:10 VAX-11 Bliss-32 V4.0-742
14-Sep-1984 12:16:53 [DEBUG.SRC]DBGEVENT.B32;1

Page 152
(17)

```
: 5459 5576 10
: 5460 5577 9
: 5461 5578 10
: 5462 5579 10
: 5463 5580 10
: 5464 5581 10
: 5465 5582 10
: 5466 5583 10
: 5467 5584 10
: 5468 5585 10
: 5469 5586 10
: 5470 5587 10
: 5471 5588 10
: 5472 5589 10
: 5473 5590 10
: 5474 5591 10
: 5475 5592 10
: 5476 5593 10
: 5477 5594 10
: 5478 5595 10
: 5479 5596 10
: 5480 5597 10
: 5481 5598 10
: 5482 5599 10
: 5483 5600 10
: 5484 5601 10
: 5485 5602 10
: 5486 5603 10
: 5487 5604 10
: 5488 5605 10
: 5489 5606 10
: 5490 5607 10
: 5491 5608 10
: 5492 5609 10
: 5493 5610 10
: 5494 5611 10
: 5495 5612 10
: 5496 5613 10
: 5497 5614 10
: 5498 5615 10
: 5499 5616 10
: 5500 5617 10
: 5501 5618 10
: 5502 5619 10
: 5503 5620 10
: 5504 5621 10
: 5505 5622 9
: 5506 5623 8
: 5507 5624 8
: 5508 5625 8
: 5509 5626 8
: 5510 5627 8
: 5511 5628 8
: 5512 5629 8
: 5513 5630 8
: 5514 5631 9
: 5515 5632 9
```

```
(.EXCEPTION_TYPE NEQ SSS_ROPRAND)
THEN
  BEGIN

    ! The event is active.
    ACTIVE_EVENT = TRUE;

    ! Get a new event descriptor
    ! and link it into the
    ! command queue. Note that
    ! this entry is placed in
    ! the queue just in front
    ! of its parent, and should
    ! be encountered first !!!
    SKIPS_ENTRY =
      DBG$GET_MEMORY
        (EVENT$K_EVENT_DESCRIPTOR_SIZE);
    INSQUE (.SKIPS_ENTRY,
      .EVENT[EVENT$L_CMD_BLINK]
    );

    ! Setup this entry to be a
    ! /RETURN skip-event,
    ! linking to the event
    ! entry causing this mess.
    SKIPS_ENTRY[EVENT$B_CMD_TYPE] =
      EVENT$K_TYPE_SKIPS;
    SKIPS_ENTRY[EVENT$B_CMD_KIND] =
      EVENT$K_KIND_ACC;
    SKIPS_ENTRY[EVENT$B_SUB_KIND] =
      EVENT$K_ACC_RTRN;
    SKIPS_ENTRY[EVENT$V_ONCE_ONLY] =
      TRUE;
    INSQUE1 (.SKIPS_ENTRY, .EVENT);

    ! Initialize the FP to 0 so it gets
    ! setup when activated.
    SKIPS_ENTRY[EVENT$L_USERS_FP] = 0;
    END;

    ! [BREAK!TRACE/RETURN skips and
    ! STEP/RETURN
    ! [EVENT$K_TYPE_SKIPS,
    ! EVENT$K_TYPE_STEPS]:
    BEGIN
      SELECTONE .EXCEPTION_TYPE OF
```



```
.. 5516      5633  9
.. 5517      5634  9
.. 5518      5635 10
.. 5519      5636 10
.. 5520      5637 10
.. 5521      5638 10
.. 5522      5639 10
.. 5523      5640 10
.. 5524      5641 10
.. 5525      5642 10
.. 5526      5643 10
.. 5527      5644 10
.. 5528      5645 10
.. 5529      5646 10
.. 5530      5647 11
.. 5531      5648 11
.. 5532      5649 11
.. 5533      5650 11
.. 5534      5651 11
.. 5535      5652 11
.. 5536      5653 11
.. 5537      5654 11
.. 5538      5655 11
.. 5539      5656 11
.. 5540      5657 12
.. 5541      5658 12
.. 5542      5659 12
.. 5543      5660 12
.. 5544      5661 12
.. 5545      5662 12
.. 5546      5663 12
.. 5547      5664 12
.. 5548      5665 12
.. 5549      5666 12
.. 5550      5667 11
.. 5551      5668 11
.. 5552      5669 11
.. 5553      5670 11
.. 5554      5671 11
.. 5555      5672 11
.. 5556      5673 10
.. 5557      5674 10
.. 5558      5675  9
.. 5559      5676  9
.. 5560      5677  9
.. 5561      5678  9
.. 5562      5679  9
.. 5563      5680  9
.. 5564      5681  9
.. 5565      5682  9
.. 5566      5683  9
.. 5567      5684  8
.. 5568      5685  8
.. 5569      5686  8
.. 5570      5687  8
.. 5571      5688  8
.. 5572      5689  8
```

```
SET
[SS$ ROPRAND]:
BEGIN
LOCAL
RIGHT_FP_FLAG;
IF .OPCODE EQL 'XX'04'
THEN
RIGHT_FP_FLAG = MATCH_RIGHT_CALL_FRAME(.EVE
ELSE
RIGHT_FP_FLAG = FALSE;
IF .RIGHT_FP_FLAG
THEN
BEGIN
ACTIVE_EVENT = TRUE;

! We've come to the RET instruction that we
! in the process of stepping over a routine.
! want to change our RETURN event back into
! ordinary STEP event, to resume stepping no
IF .EVENT [EVENT$V_STEP_BPT]
THEN
BEGIN
EVENT [EVENT$V_STEP_BPT] = FALSE;
EVENT [EVENT$B_CMD_KIND] = EVENT$K_KIND
EVENT [EVENT$B_SUB_KIND] = .EVENT [EVENT
END

! Process the entry. If on return we have b
! preempted by a GO, STEP, CONTINUE, or E
! we will stop processing.
ELSE IF DBG$PROCESS_EVENT
(.EVENT,
.SIGNAL_ARG_PTR)
THEN
EXITLOOP;
END;
END;

[SS$ TBIT]:
IF .EVENT [EVENT$V_STEP_BPT]
THEN ACTIVE_EVENT = TRUE;

[OTHERWISE]:
0;
TES;
END;

! *****SSI
! We are examining the SSI entry was built
! at DBG$SS_INT signal...
```


DBGEVENT
V04-000

H 1
16-Sep-1984 00:59:10 VAX-11 Bliss-32 V4.0-742
14-Sep-1984 12:16:53 [DEBUG.SRC]DBGEVENT.B32;1

Page 154
(17)

```
: 5573      5690      8
: 5574      5691      9
: 5575      5692      9
: 5576      5693      9
: 5577      5694      9
: 5578      5695      9
: 5579      5696     10
: 5580      5697     10
: 5581      5698     10
: 5582      5699      9
: 5583      5700     10
: 5584      5701     10
: 5585      5702     10
: 5586      5703     10
: 5587      5704     10
: 5588      5705     10
: 5589      5706      9
: 5590      5707      8
: 5591      5708      8
: 5592      5709      7
: 5593      5710      7
: 5594      5711      7
: 5595      5712      7
: 5596      5713      7
: 5597      5714      8
: 5598      5715      8
: 5599      5716      8
: 5600      5717      8
: 5601      5718      8
: 5602      5719      8
: 5603      5720      8
: 5604      5721      9
: 5605      5722      9
: 5606      5723      9
: 5607      5724      9
: 5608      5725      9
: 5609      5726      9
: 5610      5727      9
: 5611      5728      9
: 5612      5729      9
: 5613      5730      9
: 5614      5731      9
: 5615      5732      9
: 5616      5733      9
: 5617      5734      9
: 5618      5735      9
: 5619      5736      9
: 5620      5737      9
: 5621      5738      9
: 5622      5739      9
: 5623      5740      9
: 5624      5741      9
: 5625      5742      9
: 5626      5743      9
: 5627      5744      9
: 5628      5745      9
: 5629      5746      9
```

```
[EVENT$K_TYPE_SSINT]:  
BEGIN
```

```
! Well? Caused from Reserved Operand  
! fault?
```

```
IF (.DBG$RUNFRAME [DBG$L_USER_FP] EQLA  
    .EVENT [EVENT$L_USERS_FP])  
AND (.OPCODE EQLU %X'04')  
THEN
```

```
    BEGIN  
    EVENT [EVENT$V_DELETED] = TRUE;  
    ACTIVE_EVENT = TRUE;  
    EVENT$B_STOP_USER = FALSE;  
    DBG$GB_TAKE_CMD = FALSE;  
    EXITLOOP;  
    END;
```

```
END;
```

```
TES;
```

```
END;
```

```
TES;
```

```
[EVENT$K_KIND_INS]:  
BEGIN
```

```
! If this is a TBIT (or DEBUG) or BREAK  
! exception when stepping over, process  
! a step.
```

```
IF (SELECTONE .EXCEPTION_TYPE OF
```

```
    SET  
    [SS$ TBIT, SS$ DEBUG]:  
    IF .EVENT [EVENT$V_STEP_BPT]  
    THEN
```

```
        IF .EVENT [EVENT$L_ADDRESS] EQLA .USERS_PC  
        THEN  
            TRUE  
        ELSE  
            FALSE
```

```
    ELSE  
        TRUE;
```

```
[SS$ BREAK]:  
    IF .EVENT [EVENT$V_STEP_BPT]  
    THEN
```

```
        IF .EVENT [EVENT$L_ADDRESS] EQLA .USERS_PC  
        THEN  
            TRUE  
        ELSE  
            FALSE
```

```
    ELSE  
        FALSE;
```

```
[OTHERWISE]:  
    FALSE;
```

```
TES
```

```
,
```



```
: 5630      5747      8
: 5631      5748      9
: 5632      5749      9
: 5633      5750      9
: 5634      5751      9
: 5635      5752      9
: 5636      5753      9
: 5637      5754      9
: 5638      5755      9
: 5639      5756      9
: 5640      5757      9
: 5641      5758      9
: 5642      5759      9
: 5643      5760      9
: 5644      5761      9
: 5645      5762      9
: 5646      5763      9
: 5647      5764      9
: 5648      5765      9
: 5649      5766      9
: 5650      5767      9
: 5651      5768      9
: 5652      5769      9
: 5653      5770      9
: 5654      5771     10
: 5655      5772     10
: 5656      5773      9
: 5657      5774     10
: 5658      5775     10
: 5659      5776     10
: 5660      5777     10
: 5661      5778     10
: 5662      5779     10
: 5663      5780     11
: 5664      5781     11
: 5665      5782     11
: 5666      5783     11
: 5667      5784     11
: 5668      5785     11
: 5669      5786     11
: 5670      5787     11
: 5671      5788     11
: 5672      5789     10
: 5673      5790     10
: 5674      5791     10
: 5675      5792     10
: 5676      5793     10
: 5677      5794     10
: 5678      5795     10
: 5679      5796     10
: 5680      5797     10
: 5681      5798     10
: 5682      5799     10
: 5683      5800     10
: 5684      5801     10
: 5685      5802     10
: 5686      5803     12
```

```
THEN
BEGIN
```

```
! The event is active, regardless.
```

```
ACTIVE_EVENT = TRUE;
```

```
! If this was a BPT exception, turn off the
! STEP_BPT flag - we're done stepping over.
```

```
IF .EXCEPTION_TYPE EQLU SSS_BREAK
```

```
THEN
    EVENT [EVENT$V_STEP_BPT] = FALSE;
```

```
! Depending on the step type (line/instruction) process the step.
! We also check here that either we are stepping /system, or else we
! not be in system space. That is, if we find that we are stepping n
! and that we are indeed in system space, then the IF condition will
! false and we will not process the event. We actually use ADDRESS G
! as our determination of what is system space, since most system se
! are actually entered through P1 space now.
```

```
IF ((NOT .EVENT [EVENT$V_STEP_NOSYSTEM]) OR
    (.DBG$RUNFRAME[DBG$USER_PC] LSS P1_SPACE))
```

```
    AND
    (SELECTONE .EVENT [EVENT$B_SUB_KIND] OF
    SET
    [EVENT$K_INS_EVERY]:
    TRUE;
```

```
[EVENT$K_INS_CALL, EVENT$K_INS_BRAN, EVENT$K_INS_USER]:
```

```
    BEGIN
    BIND
```

```
        OP_MAP = EVENT[EVENT$L_OPCODE_LIST] : REF BITVECTOR
```

```
! Check the opcode bitmap to see if the current
! instruction is one we're looking for.
```

```
    .OP_MAP [.OPCODE]
```

```
    END;
```

```
[EVENT$K_INS_LINE]:
```

```
! In the following parenthesized expression,
! TRUE => process this event, which generally means
! stop tbit-ing and take user commands,
! FALSE => don't take this event,
! which generally means keep tbit-ing.
```

```
! If we are still in the range given by
! LO_PC, HI_PC then don't take the event -
! keep tbit-ing.
```

```
(IF (.USERS_PC GEQA .EVENT [EVENT$L_STEP_LO_PC])
```



```
: 5687 5804 11
: 5688 5805 12
: 5689 5806 11
: 5690 5807 11
: 5691 5808 12
: 5692 5809 12
: 5693 5810 12
: 5694 5811 12
: 5695 5812 12
: 5696 5813 12
: 5697 5814 12
: 5698 5815 12
: 5699 5816 12
: 5700 5817 12
: 5701 5818 12
: 5702 5819 12
: 5703 5820 12
: 5704 5821 12
: 5705 5822 12
: 5706 5823 12
: 5707 5824 12
: 5708 5825 12
: 5709 5826 12
: 5710 5827 12
: 5711 5828 12
: 5712 5829 12
: 5713 5830 12
: 5714 5831 12
: 5715 5832 13
: 5716 5833 13
: 5717 5834 13
: 5718 5835 12
: 5719 5836 12
: 5720 5837 12
: 5721 5838 12
: 5722 5839 12
: 5723 5840 12
: 5724 5841 12
: 5725 5842 12
: 5726 5843 12
: 5727 5844 12
: 5728 5845 12
: 5729 5846 12
: 5730 5847 12
: 5731 5848 12
: 5732 5849 12
: 5733 5850 12
: 5734 5851 12
: 5735 5852 12
: 5736 5853 13
: 5737 5854 10
: 5738 5855 10
: 5739 5856 10
: 5740 5857 9
: 5741 5858 9
: 5742 5859 9
: 5743 5860 9
```

```
AND
(.USERS_PC LEQA .EVENT [EVENT$$_STEP_HI_PC])
THEN FALSE
ELSE
BEGIN
LOCAL
LINENO,
MODRST,
STATUS,
STMTNO;

! The PC is out of the LO_PC, HI_PC range.
! Call the PC lookup routine to get a new range.
! If the PC lookup routine returns FALSE
! (0 or 2) then the lookup failed. In this
! case re-initialize the range to 1,0 so
! that we immediately are out of the range
! again after the next tbit.
MODRST = 0;
STATUS = DBG$PC_TO_LINE_LOOKUP(
.USERS_PC,
LINENO,
STMTNO,
EVENT[EVENT$$_STEP_LO_PC],
EVENT[EVENT$$_STEP_HI_PC],
MODRST);
IF NOT STATUS THEN
BEGIN
EVENT[EVENT$$_STEP_LO_PC] = 1;
EVENT[EVENT$$_STEP_HI_PC] = 0;
END;

! The lookup routine can return any of:
! 0 - no line number information for module
! 1 - exact match to line
! 2 - have pc/line tables, but no match for this PC
! 3 - found line, not exact match.
! We definitely want to stop and take user commands
! for 0 (examples are stepping into RTL or module
! that is not set).
! We also want to stop for 1 (stepped to the beginni
! of a new line), and 3 (stepped to the middle of
! a new line).
! For 2, we keep going - we have stepped to a
! between-lines location. RPG generates this kind
! of code, and doesn't want us to stop at
! addresses without line numbers.
(.STATUS NEQ 2)
END);

TES
)
THEN

! The entry is active.
```



```
: 5744 5861 9
: 5745 5862 10
: 5746 5863 10
: 5747 5864 10
: 5748 5865 10
: 5749 5866 10
: 5750 5867 10
: 5751 5868 10
: 5752 5869 10
: 5753 5870 10
: 5754 5871 10
: 5755 5872 10
: 5756 5873 10
: 5757 5874 10
: 5758 5875 10
: 5759 5876 9
: 5760 5877 9
: 5761 5878 8
: 5762 5879 8
: 5763 5880 7
: 5764 5881 7
: 5765 5882 7
: 5766 5883 7
: 5767 5884 7
: 5768 5885 7
: 5769 5886 7
: 5770 5887 7
: 5771 5888 7
: 5772 5889 7
: 5773 5890 7
: 5774 5891 7
: 5775 5892 7
: 5776 5893 7
: 5777 5894 7
: 5778 5895 7
: 5779 5896 6
: 5780 5897 6
: 5781 5898 5
: 5782 5899 5
: 5783 5900 5
: 5784 5901 5
: 5785 5902 5
: 5786 5903 5
: 5787 5904 5
: 5788 5905 5
: 5789 5906 5
: 5790 5907 5
: 5791 5908 5
: 5792 5909 5
: 5793 5910 5
: 5794 5911 4
: 5795 5912 4
: 5796 5913 4
: 5797 5914 4
: 5798 5915 4
: 5799 5916 4
: 5800 5917 4
```

```
!
! BEGIN
! Process the entry. If on return we
! have been preempted by a GO, STEP,
! CONTINUE, or EXIT, we will stop
! processing.
! IF DBG$PROCESS_EVENT
!     (.EVENT,
!     .SIGNAL_ARG_PTR
! )
! THEN
!     EXITLOOP;
! END;
END;
END;
[EVENT$K_KIND_EXC]:
0;
TES;
! Release any temporary memory used during the
! processing of this event. Also release all
! temporary RST Entries that have zero reference
! counts.
DBG$REL_TEMPMEM ();
DBG$RST_TEMP_RELEASE();
END;
END;
! If this entry was the 'last' one, exit this loop. Any others
! on the list now came from 'SET BREAK ...' commands in some
! entry's DO command list....
IF .EVENT EQLA .LAST1 THEN EXITLOOP;
! Link to the next entry.
! EVENT = .EVENT [EVENT$L_CMD_FLINK];
END;
! We're not skipping ACCVIO's any longer.
SKIP_ACCVIO = FALSE;
SKIP_WATCHES = FALSE;
```

```
5801
5802
5803
5804
5805
5806
5807
5808
5809
5810
5811
5812
5813
5814
5815
5816
5817
5818
5819
5820
5821
5822
5823
5824
5825
5826
5827
5828
5829
5830
5831
5832
5833
5834
5835
5836
5837
5838
5839
5840
5841
5842
5843
5844
5845
5846
5847
5848
5849
5850
5851
5852
5853
5854
5855
5856
5857
```

```
5918
5919
5920
5921
5922
5923
5924
5925
5926
5927
5928
5929
5930
5931
5932
5933
5934
5935
5936
5937
5938
5939
5940
5941
5942
5943
5944
5945
5946
5947
5948
5949
5950
5951
5952
5953
5954
5955
5956
5957
5958
5959
5960
5961
5962
5963
5964
5965
5966
5967
5968
5969
5970
5971
5972
5973
5974
```

END;

[SS\$_ACCVIO]:

```
! DEBUG uses ACCVIO's for /MODIFY events. When an ACCVIO occurs,
! we search the page queue for a page address match. If one is
! found, we flag the event as active, flag that we're skipping
! the ACCVIO, and we're done.
BEGIN
LOCAL
PAGE_ENTRY : REF EVENT$PAGE_DESCRIPTOR, ! Page entry
PAGE_ADDRESS;

IF NOT .SKIP_ACCVIOS
THEN
BEGIN

! Get the page address, and point to the 1st page entry.
PAGE_ADDRESS = .SIGNAL_ARG_PTR [12,0,32,0] AND %X'FFFFFFE0';
PAGE_ENTRY = .EVENT$PAGE_QUEUE [L_QUEUE_FLINK];

! Search the page queue for an entry with this address.
WHILE .PAGE_ENTRY NEQA EVENT$PAGE_QUEUE DO
BEGIN
IF .PAGE_ENTRY [EVENT$L_PAGE_ADDRESS] EQLA .PAGE_ADDRESS
THEN

! Found the matching entry, so flag that we're skipping
! 'cause of an ACCVIO, and exit the search loop.
BEGIN
SKIP_ACCVIOS = TRUE;
ACTIVE_EVENT = TRUE;
EXITLOOP;
END

ELSE
PAGE_ENTRY = .PAGE_ENTRY [EVENT$L_PAGE_FLINK];

END;

! Save the current PC for announcing where we were after we've
! finished executing this instruction.
ACCVIO_PC = .USERS_PC;
END;

END;
```



```
5858
5859
5860
5861
5862
5863
5864
5865
5866
5867
5868
5869
5870
5871
5872
5873
5874
5875
5876
5877
5878
5879
5880
5881
5882
5883
5884
5885
5886
5887
5888
5889
5890
5891
5892
5893
5894
5895
5896
5897
5898
5899
5900
5901
5902
5903
5904
5905
5906
5907
5908
5909
5910
5911
5912
5913
5914
```

```
5975
5976
5977
5978
5979
5980
5981
5982
5983
5984
5985
5986
5987
5988
5989
5990
5991
5992
5993
5994
5995
5996
5997
5998
5999
6000
6001
6002
6003
6004
6005
6006
6007
6008
6009
6010
6011
6012
6013
6014
6015
6016
6017
6018
6019
6020
6021
6022
6023
6024
6025
6026
6027
6028
6029
6030
6031
```

```
TES;

! If no active events were discovered, look through the queue (as we
! remember it, sans any new additions) for an exception break. If
! found process it.
IF NOT .ACTIVE_EVENT
THEN
  BEGIN

    ! For each event entry in the queue, check to see if it is active....
    EVENT = .EVENT$CMD_QUEUE [L_QUEUE_FLINK];
    WHILE .EVENT NEQA EVENT$CMD_QUEUE[A_QUEUE_ENTRY] DO
      BEGIN

        ! Make sure that the entry isn't deleted but not removed....
        IF NOT .EVENT [EVENT$V_DELETED]
        THEN

          ! If this is an exception break....
          IF .EVENT [EVENT$B_CMD_KIND] EQLU EVENT$K_KIND_EXC
          THEN
            BEGIN

              ! The event is active.
              ACTIVE_EVENT = TRUE;

              ! This is an exception break or trace, so flag it.
              DBG$GB_EXC_BRE_FLAG = TRUE;

              ! Process the event, and scoot (it's the only one).
              DBG$PROCESS_EVENT (.EVENT, .SIGNAL_ARG_PTR);
              EXITLOOP;
              END;

            ! If this entry was the 'last' one, exit this loop. Any others
            ! on the list now came from "SET BREAK ..." commands in some
            ! entry's DO command list....
            IF .EVENT EQLA .LAST1 THEN EXITLOOP;
```



```
: 5915      6032      5
: 5916      6033      5
: 5917      6034      5
: 5918      6035      4
: 5919      6036      4
: 5920      6037      3
: 5921      6038      3
: 5922      6039      2
: 5923      6040      2
: 5924      6041      2
: 5925      6042      2
: 5926      6043      2
: 5927      6044      2
: 5928      6045      2
: 5929      6046      2
: 5930      6047      2
: 5931      6048      2
: 5932      6049      2
: 5933      6050      3
: 5934      6051      3
: 5935      6052      3
: 5936      6053      3
: 5937      6054      3
: 5938      6055      3
: 5939      6056      3
: 5940      6057      3
: 5941      6058      3
: 5942      6059      3
: 5943      6060      3
: 5944      6061      3
: 5945      6062      4
: 5946      6063      4
: 5947      6064      4
: 5948      6065      4
: 5949      6066      4
: 5950      6067      4
: 5951      6068      4
: 5952      6069      4
: 5953      6070      4
: 5954      6071      4
: 5955      6072      4
: 5956      6073      4
: 5957      6074      4
: 5958      6075      4
: 5959      6076      4
: 5960      6077      4
: 5961      6078      4
: 5962      6079      4
: 5963      6080      4
: 5964      6081      4
: 5965      6082      4
: 5966      6083      4
: 5967      6084      4
: 5968      6085      4
: 5969      6086      4
: 5970      6087      4
: 5971      6088      4
```

```
! Link to the next entry.
EVENT = .EVENT [EVENT$$_CMD_FLINK];
END;

END;

END;

! If an active event was found, meaning we understood the exception,
! return check to see if a break/step ended (and the user didn't go
! or step in some DO clause). If this is the case, go back to the
! user for commands, otherwise continue. If an active event wasn't
! found, return $$$_RESIGNAL....
IF .ACTIVE_EVENT
THEN
BEGIN

! If this was not a trace event, then the EVENT$$_STOP_USER flag will
! still be on, indicating that we want to accept DEBUG commands.
! Also, if the user has typed ctrl/Y, DEBUG, then the STOP flag
! will have been turned on, and in this case we also want to take
! user commands.
IF .EVENT$$_STOP_USER OR
.DBG$$_GV_CONTROL[DBG$$_V_CONTROL_STOP]
THEN
BEGIN
LOCAL
START_PC : REF VECTOR [, WORD];

! Turn off RMS cancel Control-O bit so that user can toggle
! Control-O in the normal fashion.
DBG$$_GL_OUTPRAB [RAB$$_V_CCO] = FALSE;

! Mark that the user program has executed so that all screen
! displays will be updated before we take the next command from
! the user terminal. Note that this is suppressed if we just
! returned normally from a CALL command routine. Hence the CALL
! command does not update screen displays. If we are about to
! take commands, we also clear the CALL command status flag.
IF .DBG$$_GB_CALL_NORMAL_RET NEQ 2
THEN
DBG$$_GV_CONTROL[DBG$$_V_CONTROL_SCREEN] = TRUE;

IF .DBG$$_GB_TAKE_CMD THEN DBG$$_GB_CALL_NORMAL_RET = 0;

! As long as no commands in the any processed DO clause caused the
! program to continue (or restart), the command taking flag will
```



```

: 5972      6089  4      ! still be turned on, and further commands can be processed from
: 5973      6090  4      ! the input device. Otherwise, the while loop below does not
: 5974      6091  4      ! execute at all.
: 5975      6092  4      !
: 5976      6093  4      WHILE .DBG$GB TAKE CMD DO
: 5977      6094  4          DBG$COMMAND_PROC ();
: 5978      6095  4
: 5979      6096  4      END;
: 5980      6097  4
: 5981      6098  4
: 5982      6099  4      ! Activate events....
: 5983      6100  4      DBG$ACTIVATE_EVENTS ();
: 5984      6101  4
: 5985      6102  4
: 5986      6103  4
: 5987      6104  4      ! Return the proper status, depending primarily on whether this was
: 5988      6105  4      ! an EXCEPTION break/trace, or a normal type.
: 5989      6106  4
: 5990      6107  4      IF .DBG$GB_EXC_BRE_FLAG
: 5991      6108  4      THEN
: 5992      6109  4          BEGIN
: 5993      6110  4              DBG$RUNFRAME[DBG$V_TBIT] = .ORIG_TBIT;
: 5994      6111  4
: 5995      6112  4
: 5996      6113  4              ! Resignal the exception so that handlers can get it
: 5997      6114  4              ! (unless the user said GO <address> to go somewhere
: 5998      6115  4              ! else).
: 5999      6116  4
: 6000      6117  4              IF NOT .DBG$GB_TAKE_CMD
: 6001      6118  4              THEN
: 6002      6119  4                  IF .DBG$GB_GO_ARG_FLAG
: 6003      6120  4                  THEN
: 6004      6121  4                      RETURN SS$_CONTINUE
: 6005      6122  4                  ELSE
: 6006      6123  4                      RETURN SS$_RESIGNAL
: 6007      6124  4                  ELSE
: 6008      6125  4                      RETURN SS$_RESIGNAL
: 6009      6126  4                  END
: 6010      6127  4              ELSE
: 6011      6128  4
: 6012      6129  4
: 6013      6130  4              ! Return to the user.
: 6014      6131  4              RETURN SS$_CONTINUE;
: 6015      6132  4          END;
: 6016      6133  4
: 6017      6134  4
: 6018      6135  4
: 6019      6136  4      ! Didn't find an active event, so resignal the exception....
: 6020      6137  4      !
: 6021      6138  4      DBG$ACTIVATE_EVENTS ();
: 6022      6139  4      DBG$RUNFRAME[DBG$V_TBIT] = .ORIG_TBIT;
: 6023      6140  4      RETURN SS$_RESIGNAL;
: 6024      6141  4      END;
```


				OFFC 00000	.ENTRY	DBG\$EXCEPTION HANDLER, Save R2,R3,R4,R5,R6,-, R7,R8,R9,R10,R11	
04	AE	00000000G	00	24 C2 00002	SUBL2	#36, SP	5006
			01	04 EF 00005	EXTZV	#4, #1, DBG\$RUNFRAME+68, ORIG_TBIT	5099
			52 04 AC D0 0000F	MOVL	SIGNAL_ARG_PTR, R2		5104
			5A 04 A2 D0 00013	MOVL	4(R2), EXCEPTION_TYPE		
			5B 00000000G 00 D0 00017	MOVL	DBG\$RUNFRAME+64, USERS_PC		5109
	6B		01 00 0C 0001E	PROBER	#0, #1, (USERS_PC)		5110
			04 12 00022	BNEQ	1\$		
			6E D4 00024	CLRL	OPCODE		
			1D 11 00026	BRB	2\$		
			6E 6B 9A 00028 1\$:	MOVZBL	(USERS_PC), OPCODE		5111
	000000FD		8F 6E D1 0002B	CMPL	OPCODE, #253		
			11 1F 00032	BLSSU	2\$		
	6B		02 00 0C 00034	PROBER	#0, #2, (USERS_PC)		5112
			0B 13 00038	BEQL	2\$		
			6E 01 AB 9A 0003A	MOVZBL	1(USERS_PC), OPCODE		5113
			6E 00000100 8F C0 0003E	ADDL2	#256, OPCODE		
	00000464		8F 5A D1 0C045 2\$:	CMPL	EXCEPTION_TYPE, #1124		5119
			4D 12 0004C	BNEQ	4\$		
			50 00000000G 00 9E 0004E	MOVAB	DBG\$PSEUDO_EXIT, R0		5122
			50 5B D1 00055	CMPL	USERS_PC, R0		
			30 13 00058	BEQL	3\$		
			50 00000000G 00 9E 0005A	MOVAB	DBG\$TERM_HANDLR, R0		5123
			50 5B D1 00061	CMPL	USERS_PC, R0		
			24 13 00064	BEQL	3\$		
			50 00000000G 00 9E 00066	MOVAB	DBG\$USER_EXIT, R0		5124
			50 5B D1 0006D	CMPL	USERS_PC, R0		
			18 13 00070	BEQL	3\$		
			50 00000000G 00 9E 00072	MOVAB	PRIM_HANDL_2, R0		5125
			50 5B D1 00079	CMPL	USERS_PC, R0		
			0C 13 0007C	BEQL	3\$		
			50 00000000G 00 9E 0007E	MOVAB	DBG\$PSEUDO_SSI, R0		5126
			50 5B D1 00085	CMPL	USERS_PC, R0		
			11 12 00088	BNEQ	4\$		
	00000000G		00 10 8A 0008A 3\$:	BICB2	#16, DBG\$RUNFRAME+68		5129
	00000000G		00 80 8F 88 00091	BISB2	#128, DBG\$GL_OUTPRAB+7		5130
			18 11 00099	BRB	5\$		5131
	000006A1		8F 5A D1 0009B 4\$:	CMPL	EXCEPTION_TYPE, #1697		5139
			12 12 000A2	BNEQ	6\$		
	FE12	CF	00 FB 000A4	CALLS	#0, DBG\$DEACTIVATE_EVENTS		5154
	OC	B2	08 B2 90 000A9	MOVB	28(R2), 212(R2)		5155
	FA4B	CF	00 FB 000AE	CALLS	#0, DBG\$ACTIVATE_EVENTS		5156
			04FA 31 000B3 5\$:	BRW	71\$		5157
	00028793		8F 5A D1 000B6 6\$:	CMPL	EXCEPTION_TYPE, #165779		5169
			03 13 000BD	BEQL	7\$		
			00B6 31 000BF	BRW	11\$		
			50 08 A2 9E 000C2 7\$:	MOVAB	8(R2), SIGARG		5186
			54 14 A0 D0 000C6	MOVL	20(SIGARG), BIT_15_CNT_SSV		5192
			01 64 91 000CA	CMPB	(BIT_15_CNT_SSV), #1		5193
			E4 1A 000CD	BGTRU	5\$		
			53 10 A0 D0 000CF	MOVL	16(SIGARG), BIT_15_CNT		5200
			63 95 000D3	TSTB	(BIT_15_CNT)		5201
			DC 12 000D5	BNEQ	5\$		
	00000000'		EF 01 90 000D7	MOVB	#1, DBG\$GB_SET_WATCH_FLAG		5206
			52 OC A0 D0 000DE	MOVL	12(SIGARG), USERS_FP		5207

56	00000000'	EF	D0	000E2	MOVL	EVENT\$CMD_QUEUE, EVENT	5208
		10	DD	000E9	PUSHL	#16	5209
00000000G	00	01	FB	000EB	CALLS	#1, DBG\$GET_MEMORY	
04	B6	60	0E	000F2	INSQUE	(SSINT_ENTRY), @4(EVENT)	5210
14	A0	06	B0	000F6	MOVW	#6, 20(SSINT_ENTRY)	5211
16	A0	04	90	000FA	MOVB	#4, 22(SSINT_ENTRY)	5213
17	A0	04	88	000FE	BISB2	#4, 23(SSINT_ENTRY)	5214
08	A0	50	D0	00102	MOVL	SSINT_ENTRY, 8(SSINT_ENTRY)	5215
0C	A0	50	D0	00106	MOVL	SSINT_ENTRY, 12(SSINT_ENTRY)	5216
24	A0	52	D0	0010A	MOVL	USERS_FP, 36(SSINT_ENTRY)	5217
05	A2	80	8F	88	BISB2	#128, 5(USERS_FP)	5218
1C	A0	54	D0	00113	MOVL	BIT_15_CNT_SS0, 28(SSINT_ENTRY)	5219
2C	A0	53	D0	00117	MOVL	BIT_15_CNT, 44(SSINT_ENTRY)	5220
20	A0	A2	D0	0011B	MOVL	16(USERS_FP), 32(SSINT_ENTRY)	5221
00000000'	EF	10	A2	D0	MOVL	16(USERS_FP), WATCH_PC	5222
			63	96	INCB	(BIT_15_CNT)	5223
			63	91	CMPB	(BIT_15_CNT), #1	5224
01			84	12	BNEQ	5\$	
52	00000000'	EF	D0	0012F	MOVL	EVENT\$PAGE_QUEUE, PAGE_ENTRY	5227
50	00000000'	EF	9E	00136	MOVAB	EVENT\$PAGE_QUEUE, R0	5228
50			52	D1	CMPL	PAGE_ENTRY, R0	
			03	12	BNEQ	9\$	
			046B	31	BRW	71\$	
1C	AE	08	A2	D0	MOVL	8(PAGE_ENTRY), ADDRESS	5230
20	AE	08	A2	D0	MOVL	8(PAGE_ENTRY), ADDRESS+4	5231
			7E	D4	CLRL	-(SP)	5234
	7E	0E	A2	9A	MOVZBL	14(PAGE_ENTRY), -(SP)	
			7E	7C	CLRQ	-(SP)	
		2C	AE	9F	PUSHAB	ADDRESS	
00000000G	00	05	FB	0015A	CALLS	#5, SYS\$SETPRT	
	0F	50	E8	00161	BLBS	R0, 10\$	
00000000G	00	000284C4	8F	DD	PUSHL	#165060	5236
			01	FB	CALLS	#1, LIB\$SIGNAL	
			C3	11	BRB	8\$	
	52		62	D0	MOVL	(PAGE_ENTRY), PAGE_ENTRY	5238
			BE	11	BRB	8\$	5228
	00000000G	00	94	00178	CLRB	DBG\$GB_EXC_BRE_FLAG	5248
			59	D4	CLRL	ACTIVE_EVENT	5254
00000000'	EF	01	D0	00180	MOVL	#1, TYPE_SOURCE	5260
	00000000'	EF	94	00187	CLRB	EVENT\$B_STOP_USER	5267
00000000G	00	01	90	0018D	MOVB	#1, DBG\$GB_TAKE_CMD	5274
FD22	CF	00	FB	00194	CALLS	#0, DBG\$DEACTIVATE_EVENTS	5280
0000046C	8F	5A	D1	00199	CMPL	EXCEPTION_TYPE, #1T32	5281
			0A	12	BNEQ	12\$	
00000000'	EF	01	90	001A2	MOVB	#1, EVENT\$B_STOP_USER	5295
	59	01	D0	001A9	MOVL	#1, ACTIVE_EVENT	5296
	00000000G	00	D5	001AC	TSTL	DBG\$RUNFRAME+64	5302
			26	12	BNEQ	15\$	
56	00000000'	EF	9E	001B4	MOVAB	EVENT\$CMD_QUEUE, EVENT	5305
56			66	D0	MOVL	(EVENT), EVENT	5308
50	00000000'	EF	9E	001BE	MOVAB	EVENT\$CMD_QUEUE, R0	5309
50			56	D1	CMPL	EVENT, R0	
			03	12	BNEQ	14\$	
			0378	31	BRW	65\$	
03	14	A6	91	001CD	CMPB	20(EVENT), #3	5310
			E8	12	BNEQ	13\$	
17	A6	80	8F	88	BISB2	#128, 23(EVENT)	5312

00000414	8F	E1	11	001D8	BRB	13\$	5306	
		5A	D1	001DA	CMPL	EXCEPTION_TYPE, #1044	5327	
		1E	13	001E1	BEQL	16\$		
00000454	8F	5A	D1	001E3	CMPL	EXCEPTION_TYPE, #1108		
		15	13	001EA	BEQL	16\$		
00000464	8F	5A	D1	001EC	CMPL	EXCEPTION_TYPE, #1124		
		0C	13	001F3	BEQL	16\$		
0000046C	8F	5A	D1	001F5	CMPL	EXCEPTION_TYPE, #1132		
		03	13	001FC	BEQL	16\$		
		02B8	31	001FE	BRW	58\$		
0000046C	8F	5A	D1	00201	CMPL	EXCEPTION_TYPE, #1132	5328	
		0B	12	00208	BNEQ	17\$		
03 00000000G	00	01	E0	0020A	BBS	#1, DBG\$GV_CONTROL+1, 17\$	5329	
		02EE	31	00212	BRW	62\$		
08	AE	00000000'	EF	D0	00215	MOVL	EVENT\$CMD_QUEUE+4, LAST1	5345
	56	00000000'	EF	D0	0021D	MOVL	EVENT\$CMD_QUEUE, EVENT	5351
	50	00000000'	EF	9E	00224	MOVAB	EVENT\$CMD_QUEUE, R0	5352
	50		56	D1	0022B	CMPL	EVENT, R0	
			03	12	0022E	BNEQ	19\$	
			027E	31	00230	BRW	57\$	
	58	14	A6	9E	00233	MOVAB	20(EVENT), R8	5358
			68	D5	00237	TSTL	(R8)	
			16	19	00239	BLSS	20\$	
	05		68	91	0023B	CMPB	(R8), #5	5365
			14	12	0023E	BNEQ	21\$	
03	A8	80	8F	88	00240	BISB2	#128, 3(R8)	5376
00000464	8F		5A	D1	00245	CMPL	EXCEPTION_TYPE, #1124	5381
			03	12	0024C	BNEQ	20\$	
	59		01	D0	0024E	MOVL	#1, ACTIVE_EVENT	5383
			0251	31	00251	BRW	56\$	5365
02	00	01	A8	8F	00254	CASEB	1(R8), #0, #2	5398
023E	018A		0006		00259	.WORD	23\$-22\$,-	
							40\$-22\$,-	
							55\$-22\$,-	
04	00	02	A8	8F	0025F	CASEB	2(R8), #0, #4	5406
00B6	000C	0233	0233		00264	.WORD	55\$-24\$,-	
			00C2		0026C		55\$-24\$,-	
							25\$-24\$,-	
							31\$-24\$,-	
							33\$-24\$	
			5C	11	0026E	BRB	27\$	
	0E	00000000'	EF	E8	00270	BL9S	SKIP_ACCVIOS, 26\$	5430
	07	00000000'	EF	E8	00277	BLBS	SKIP_WATCHES, 26\$	5431
	47	0C000000'	EF	E9	0027E	BLBC	DBG\$GB_SET_WATCH_FLAG, 27\$	5432
	59		01	D0	00285	MOVL	#1, ACTIVE_EVENT	5441
		34	A6	9F	00288	PUSHAB	52(EVENT)	5454
00000000G	00		01	FB	0028B	CALLS	#1, DBG\$DATA_LENGTH	
	54		50	D0	00292	MOVL	R0, LENGTH	
	50	07	A4	9E	00295	MOVAB	7(R4), R0	5455
54	50	FD	8F	78	00299	ASHL	#-3, R0, LENGTH	
			56	DD	0029E	PUSHL	EVENT	5456
	0000V	CF	01	FB	002A0	CALLS	#1, GET_WATCH_POINT_VALUE	
	00000000'	EF	A6	D0	002A5	MOVL	48(EVENT), OLDVALUE	5458
			50	D0	002AD	MOVL	OLDVALUE, R0	5467
			55	D0	002B4	MOVL	NEWVALUE, R5	5469
20	A5	20	A0	54	29	CMPC3	LENGTH, 32(R0), 32(R5)	
			0C	12	002C1	BNEQ	28\$	

			55	DD	002C3	PUSHL	R5		5473
			01	FB	002C5	CALLS	#1, DBG\$NFREE_DESC		
			01C8	31	002CC	BRW	55\$		
			EF	E8	002CF	BLBS	SKIP WATCHES, 29\$		5480
			EF	E9	002D6	BLBC	DBG\$GB SET WATCH_FLAG, 30\$		5481
			EF	D0	002DD	MOVL	WATCH_PC, ACCVIO_PC		5483
			04	AC	DD 002E8	PUSHL	SIGNAL_ARG_PTR		5486
			56	DD	002EB	PUSHL	EVENT		5485
			02	FB	002ED	CALLS	#2, DBG\$PROCESS_EVENT		
			50	D0	002F2	MOVL	R0, STATUS		
			58	EF	D0 002F6	MOVL	NEWVALUE, R8		5496
			50	EF	D0 002FD	MOVL	OLDVALUE, R0		5497
20	A0	20	A8	54	28 00304	MOVC3	LENGTH, 32(R8), 32(R0)		
			58	DD	0030A	PUSHL	R8		5499
			01	FB	0030C	CALLS	#1, DBG\$NFREE_DESC		
			B5	0C	AE	BLBC	STATUS, 27\$		5506
			5B	2C	0197	BRW	57\$		5508
			59	AC	12 0031E	CMPL	44(EVENT), USERS_PC		5524
			01	01	D0 00320	MOVL	#1, ACTIVE_EVENT		5535
			01	68	91 00326	BRW	54\$		5538
			5B	2C	3C 1A 00329	CMPB	(R8), #1		5563
			5B	2C	A6 D1 0032B	BGTRU	34\$		
			9B	12	0032F	CMPL	44(EVENT), USERS_PC		5575
			5A	D1	00331	BNEQ	27\$		
			92	13	00338	CMPL	EXCEPTION_TYPE, #1108		5576
			01	D0	0033A	BEQL	27\$		
			10	DD	0033D	MOVL	#1, ACTIVE_EVENT		5583
			01	FB	0033F	PUSHL	#16		5596
			57	D0	00346	CALLS	#1, DBG\$GET MEMORY		
			04	B6	67 0E 00349	MOVL	R0, SKIPS_ENTRY		
			14	A7	04 B0 0034D	INSQUE	(SKIPS_ENTRY), 24(EVENT)		5598
			16	A7	04 90 00351	MOVW	#4, 20(SKIPS_ENTRY)		5607
			17	A7	04 88 00355	MOVB	#4, 22(SKIPS_ENTRY)		5611
			56	DD	00359	BISB2	#4, 23(SKIPS_ENTRY)		5613
			57	DD	0035B	PUSHL	EVENT		5615
			02	FB	0035D	PUSHL	SKIPS_ENTRY		
			A7	D4	00362	CALLS	#2, INSQUE1		
			4E	11	00365	CLRL	36(SKIPS_ENTRY)		5621
			03	68	91 00367	BRB	38\$		5557
			04	4C	1F 0036A	CMPB	(R8), #3		5629
			04	68	91 0036C	BLSSU	39\$		
			47	1A	0036F	CMPB	(R8), #4		
			5A	D1	00371	BGTRU	39\$		
			2A	12	00378	CMPL	EXCEPTION_TYPE, #1108		5634
			04	6E	D1 0037A	BNEQ	37\$		
			09	12	0037D	CMPL	OPCODE, #4		5639
			56	DD	0037F	BNEQ	35\$		
			01	FB	00381	PUSHL	EVENT		5641
			02	11	00386	CALLS	#1, MATCH_RIGHT_CALL_FRAME		
			50	D4	00388	BRB	36\$		
			50	E9	0038A	CLRL	RIGHT_FP_FLAG		5643
			01	D0	0038D	BLBC	RIGHT_FP_FLAG, 38\$		5645
			02	E1	00390	MOVL	#1, ACTIVE_EVENT		5648
8E	19	A6	04	8A	00395	BBC	#2, 25(EVENT), 32\$		5655
	19	A6	01	90	00399	BICB2	#4, 25(EVENT)		5658
	01	A8	01	90	00399	MOVB	#1, 1(R8)		5659

02	A8	1B	A6	90	0039D	MOVB	27(EVENT), 2(R8)	5660	
			11	11	003A2	BRB	38\$	5655	
00000464	8F		5A	D1	003A4	37\$:	CPL	EXCEPTION_TYPE, #1124	5677
			61	12	003AB		BNEQ	45\$	
53	19	A6	02	E1	003AD		BBC	#2, 25(EVENT), 43\$	5678
		59	01	D0	003B2		MOVL	#1, ACTIVE_EVENT	5679
			00DF	31	003B5	38\$:	BRW	55\$	5678
		06	68	91	003B8	39\$:	CMPB	(R8), #6	5690
			F8	12	003BB		BNEQ	38\$	
24	A6	00000000G	00	D1	003BD		CPL	DBG\$RUNFRAME+56, 36(EVENT)	5697
			EE	12	003C5		BNEQ	38\$	
	04		6E	D1	003C7		CPL	OPCODE, #4	5698
			E9	12	003CA		BNEQ	38\$	
03	A8	80	8F	88	003CC		BISB2	#128, 3(R8)	5701
	59		01	D0	003D1		MOVL	#1, ACTIVE_EVENT	5702
		00000000'	EF	94	003D4		CLRB	EVENT\$B_STOP_USER	5703
		00000000G	00	94	003DA		CLRB	DBG\$GB_TAKE_CMD	5704
			00CE	31	003E0		BRW	57\$	5700
00000464	8F		5A	D1	003E3	40\$:	CPL	EXCEPTION_TYPE, #1124	5723
			09	13	003EA		BEQL	41\$	
0000046C	8F		5A	D1	003EC		CPL	EXCEPTION_TYPE, #1132	
			07	12	003F3		BNEQ	42\$	
16	19	A6	02	E1	003F5	41\$:	BBC	#2, 25(EVENT), 46\$	5724
			0E	11	003FA		BRB	44\$	5726
00000414	8F		5A	D1	003FC	42\$:	CPL	EXCEPTION_TYPE, #1044	5733
			B0	12	00403		BNEQ	38\$	
AB	19	A6	02	E1	00405	43\$:	BBC	#2, 25(EVENT), 38\$	5734
		5B	A6	D1	0040A	44\$:	CPL	44(EVENT), USERS_PC	5736
			A5	12	0040E	45\$:	BNEQ	38\$	
		59	01	D0	00410	46\$:	MOVL	#1, ACTIVE_EVENT	5753
00000414	8F		5A	D1	00413		CPL	EXCEPTION_TYPE, #1044	5759
			04	12	0041A		BNEQ	47\$	
	19	A6	04	8A	0041C		BICB2	#4, 25(EVENT)	5761
			A6	95	00420	47\$:	TSTB	24(EVENT)	5771
		18	0D	18	00423		BGEQ	48\$	
40000000	8F	00000000G	00	D1	00425		CPL	DBG\$RUNFRAME+64, #1073741824	5772
			65	18	00430		BGEQ	55\$	
	08	02	A8	91	00432	48\$:	CMPB	2(R8), #8	5776
			52	13	00436		BEQL	54\$	
	05	02	A8	91	00438		CMPB	2(R8), #5	5779
			06	1F	0043C		BLSSU	49\$	
	06	02	A8	91	0043E		CMPB	2(R8), #6	
			06	1B	00442		BLEQU	50\$	
	09	02	A8	91	00444	49\$:	CMPB	2(R8), #9	
			07	12	00448		BNEQ	51\$	
48	28	B6	6E	E1	0044A	50\$:	BBC	OPCODE, 240(EVENT), 55\$	5788
			39	11	0044F		BRB	54\$	
	07	02	A8	91	00451	51\$:	CMPB	2(R8), #7	5791
			33	12	00455		BNEQ	54\$	
34	A6		5B	D1	00457		CPL	USERS_PC, 52(EVENT)	5803
			06	1F	0045B		BLSSU	52\$	
38	A6		5B	D1	0045D		CPL	USERS_PC, 56(EVENT)	5805
			34	1B	00461		BLEQU	55\$	
		10	AE	D4	00463	52\$:	CLRL	MODRST	5823
		10	AE	9F	00466		PUSHAB	MODRST	5829
		38	A6	9F	00469		PUSHAB	56(EVENT)	
		34	A6	9F	0046C		PUSHAB	52(EVENT)	5828

		20	AE	9F	0046F	PUSHAB	STMTNO	5824		
		28	AE	9F	00472	PUSHAB	LINENO			
			5B	DD	00475	PUSHL	USERS_PC	5829		
00000000G	00		06	FB	00477	CALLS	#6, DBG\$PC_TO_LINE_LOOKUP			
	04		50	E8	0047E	BLBS	STATUS, 53\$	5831		
34	A6		01	7D	00481	MOVQ	#1, 52(EVENT)	5833		
	02		50	D1	00485	53\$:	CMPL	STATUS, #2	5853	
			0D	13	00488	BEQL	55\$			
		04	AC	DD	0048A	54\$:	PUSHL	SIGNAL_ARG_PTR	5871	
			56	DD	0048D	PUSHL	EVENT	5870		
0000V	CF		02	FB	0048F	CALLS	#2, DBG\$PROCESS_EVENT			
	1A		50	E8	00494	BLBS	R0, 57\$			
00000000G	00		00	FB	00497	55\$:	CALLS	#0, DBG\$REL_TEMP_MEM	5894	
00000000G	00		00	FB	0049E	CALLS	#0, DBG\$RST_TEMP_RELEASE	5895		
	08		56	D1	004A5	56\$:	CMPL	EVENT, LASTI	5905	
			06	13	004A9	BEQL	57\$			
	56		66	D0	004AB	MOVL	(EVENT), EVENT	5910		
			FD73	31	004AE	BRW	18\$	5352		
		00000000'	EF	7C	004B1	57\$:	CLRQ	SKIP_ACCVIOS	5916	
			4A	11	004B7	BRB	62\$	5328		
			5A	D1	004B9	58\$:	CMPL	EXCEPTION_TYPE, #12	5921	
	0C		45	12	004BC	BNEQ	62\$			
	3E	00000000'	EF	E8	004BE	BLBS	SKIP_ACCVIOS, 62\$	5934		
	50	04	AC	D0	004C5	MOVL	SIGNAL_ARG_PTR, R0	5941		
52	0C	A0	000001FF	8F	CB	004C9	BICL3	#511, T2(R0), PAGE_ADDRESS		
		50	00000000'	EF	D0	004D2	MOVL	EVENT\$PAGE_QUEUE, PAGE_ENTRY	5942	
		51	00000000'	EF	9E	004D9	59\$:	MOVAB	EVENT\$PAGE_QUEUE, R1	5947
		51		50	D1	004E0	CMPL	PAGE_ENTRY, R1		
			17	13	004E3	BEQL	61\$			
	52	08	A0	D1	004E5	CMPL	8(PAGE_ENTRY), PAGE_ADDRESS	5949		
			0C	12	004E9	BNEQ	60\$			
00000000'	EF		01	D0	004EB	MOVL	#1, SKIP_ACCVIOS	5957		
	59		01	D0	004F2	MOVL	#1, ACTIVE_EVENT	5958		
			05	11	004F5	BRB	61\$	5956		
	50		60	D0	004F7	60\$:	MOVL	(PAGE_ENTRY), PAGE_ENTRY	5963	
			DD	11	004FA	BRB	59\$	5947		
00000000'	EF		5B	D0	004FC	61\$:	MOVL	USERS_PC, ACCVIO_PC	5971	
	42		59	E8	00503	62\$:	BLBS	ACTIVE_EVENT, 66\$	5983	
	56	00000000'	EF	D0	00506	MOVL	EVENT\$CMD_QUEUE, EVENT	5990		
	50	00000000'	EF	9E	0050D	63\$:	MOVAB	EVENT\$CMD_QUEUE, R0	5991	
	50		56	D1	00514	CMPL	EVENT, R0			
			2C	13	00517	BEQL	65\$			
		17	A6	95	00519	TSTB	23(EVENT)	5997		
			1C	19	0051C	BLSS	64\$			
	02	15	A6	91	0051E	CMPB	21(EVENT), #2	6003		
			16	12	00522	BNEQ	64\$			
	59		01	D0	00524	MOVL	#1, ACTIVE_EVENT	6010		
00000000G	00		01	90	00527	MOVB	#1, DBG\$GB_EXC_BRE_FLAG	6015		
		04	AC	DD	0052E	PUSHL	SIGNAL_ARG_PTR	6020		
			56	DD	00531	PUSHL	EVENT			
0000V	CF		02	FB	00533	CALLS	#2, DBG\$PROCESS_EVENT			
			0B	11	00538	BRB	65\$	6005		
	08		56	D1	0053A	64\$:	CMPL	EVENT, LASTI	6029	
			05	13	0053E	BEQL	65\$			
	56		66	D0	00540	MOVL	(EVENT), EVENT	6034		
			C8	11	00543	BRB	63\$	5991		
	6C		59	E9	00545	65\$:	BLBC	ACTIVE_EVENT, 72\$	6048	

DBGEVENT
V04-000

1 2
16-Sep-1984 00:59:10
14-Sep-1984 12:16:53

VAX-11 Bliss-32 V4.0-742
[DEBUG.SRC]DBGEVENT.B32;1

Page 168
(17)

		08	00000000'	EF	E8	00548	66\$:	BLBS	EVENT\$B STOP USER, 67\$:	6059	
35	00000000G	00		01	E1	0054F		BBC	#1, DBG\$GV CONTROL+1, 70\$:	6060	
	00000000G	00	80	8F	8A	00557	67\$:	BICB2	#128, DBG\$GL OUTPRAB+7	:	6070	
		02	00000000G	00	91	0055F		CMPB	DBG\$GB_CALL_NORMAL_RET, #2	:	6080	
		07		13	00566			BEQL	68\$:		
	00000000G	00		08	88	00568		BISB2	#8, DBG\$GV CONTROL+1	:	6082	
		16	00000000G	00	E9	0056F	68\$:	BLBC	DBG\$GB_TAKE_CMD, 70\$:	6084	
			00000000G	00	94	00576		CLRB	DBG\$GB_CALL_NORMAL_RET	:		
		09	00000000G	00	E9	0057C	69\$:	BLBC	DBG\$GB_TAKE_CMD, 70\$:	6093	
	00000000G	00		00	FB	00583		CALLS	#0, DBG\$COMMAND_PROC	:	6094	
				FO	11	0058A		BRB	69\$:		
	F56D	CF		00	FB	0058C	70\$:	CALLS	#0, DBG\$ACTIVATE_EVENTS	:	6101	
		18	00000000G	00	E9	00591		BLBC	DBG\$GB_EXC_BRE_FLAG, 71\$:	6107	
00000000G	00	01		04	04	AE	FO	00598	INSV	ORIG TBIT, #4, #1, DBG\$RUNFRAME+68	:	6110
		1A	00000000G	00	E8	005A2		BLBS	DBG\$GB_TAKE_CMD, 73\$:	6117	
		13	00000000G	00	E9	005A9		BLBC	DBG\$GB_GO_ARG_FLAG, 73\$:	6119	
		50		01	D0	005B0	71\$:	MOVL	#1, R0	:	6132	
					04	005B3		RET		:		
	F545	CF		00	FB	005B4	72\$:	CALLS	#0, DBG\$ACTIVATE_EVENTS	:	6138	
00000000G	00	01		04	04	AE	FO	005B9	INSV	ORIG TBIT, #4, #T, DBG\$RUNFRAME+68	:	6139
		50	0918	8F	3C	005C3	73\$:	MOVZWL	#2328, R0	:	6140	
				04	005C8			RET		:	6141	

; Routine Size: 1481 bytes, Routine Base: DBG\$CODE + 1E6F


```
: 6026      6142 1 GLOBAL ROUTINE DBG$PROCESS_EVENT(EVENT, SIGNAL_ARG_PTR) =
: 6027      6143 1
: 6028      6144 1 FUNCTIONAL DESCRIPTION:
: 6029      6145 1
: 6030      6146 1     An eventpoint has been determined to be active. It must now be
: 6031      6147 1     processed. This includes handling any /AFTER, WHEN, and DO.
: 6032      6148 1
: 6033      6149 1
: 6034      6150 1 FORMAL PARAMETERS:
: 6035      6151 1
: 6036      6152 1     EVENT : The address of the event entry.
: 6037      6153 1
: 6038      6154 1
: 6039      6155 1 IMPLICIT OUTPUTS:
: 6040      6156 1
: 6041      6157 1     Processing the event may involve the execution of user commands.
: 6042      6158 1
: 6043      6159 1
: 6044      6160 1 ROUTINE VALUE:
: 6045      6161 1
: 6046      6162 1     TRUE is a GO, STEP, CONTINUE, or EXIT command was executed within
: 6047      6163 1     a DO clause.
: 6048      6164 1
: 6049      6165 1
: 6050      6166 1 SIDE EFFECTS:
: 6051      6167 1
: 6052      6168 1     Any number of things.
: 6053      6169 1
: 6054      6170 2 BEGIN
: 6055      6171 2
: 6056      6172 2 MAP
: 6057      6173 2     EVENT :      REF EVENT$EVENT_DESCRIPTOR, ! Event entry pointer
: 6058      6174 2     SIGNAL_ARG_PTR :  REF BLOCK[,BYTE];
: 6059      6175 2
: 6060      6176 2 BUILTIN CALLG;
: 6061      6177 2 EXTERNAL ROUTINE dbg$final_handl;
: 6062      6178 2
: 6063      6179 2 LOCAL
: 6064      6180 2     DO_LIST_ENTRY : REF EVENT$DO_LIST_DESCRIPTOR, ! DO LIST descriptor
: 6065      6181 2     MESSAGE_VECT,      ! Message Vector
: 6066      6182 2     SAVED_CISHEAD,      ! Saved pointer to DBG$GL_CISHEAD
: 6067      6183 2     SAVED_LEVEL,      ! Saved level of CIS nesting
: 6068      6184 2     EXCEPTION_TYPE;
: 6069      6185 2
: 6070      6186 2
: 6071      6187 2 ENABLE dbg$final_handl;
: 6072      6188 2
: 6073      6189 2 ! Save the exception type (name);
: 6074      6190 2 !
: 6075      6191 2 EXCEPTION_TYPE = .SIGNAL_ARG_PTR [CHF$SIG_NAME];
: 6076      6192 2
: 6077      6193 2
: 6078      6194 2 ! If this is a skips-entry, 'delete' this one and point to the parent.
: 6079      6195 2 ! It is the one we want to actually process.
: 6080      6196 2
: 6081      6197 2 IF .EVENT [EVENT$B_CMD_TYPE] EQLU EVENT$K_TYPE_SKIPS
: 6082      6198 2 THEN
```

```

6083      6199      3      BEGIN
6084      6200      EVENT [EVENT$V_DELETED] = TRUE;
6085      6201      WHILE .EVENT [EVENT$B_CMD_TYPE] EQLU EVENT$K_TYPE_SKIPS DO
6086      6202      EVENT = .EVENT [EVENT$L_EXC_FLINK];
6087      6203      END;
6088      6204
6089      6205
6090      6206      ! If this is a steps-event, process it differently from the others.
6091      6207
6092      6208      IF .EVENT [EVENT$B_CMD_TYPE] EQLU EVENT$K_TYPE_STEPS
6093      6209      THEN
6094      6210      BEGIN
6095      6211
6096      6212
6097      6213      ! Decrement the step (after) count.
6098      6214
6099      6215      IF ((EVENT [EVENT$L_AFTER_COUNT] =
6100      6216      .EVENT [EVENT$L_AFTER_COUNT] - 1
6101      6217      ) EQLU 0
6102      6218      )
6103      6219      THEN
6104      6220
6105      6221
6106      6222      ! The stepping is complete....
6107      6223
6108      6224      BEGIN
6109      6225
6110      6226      ! Special case for STEP/EXCEPTION.
6111      6227
6112      6228      IF .EVENT[EVENT$B_CMD_KIND] EQL EVENT$K_KIND_EXC
6113      6229      THEN
6114      6230      BEGIN
6115      6231
6116      6232
6117      6233      ! This is an exception break or trace, so flag it.
6118      6234
6119      6235      DBG$GB_EXC_BRE_FLAG = TRUE;
6120      6236
6121      6237
6122      6238      ! Format and output the message.
6123      6239
6124      6240      DBG$PUTMSG (.SIGNAL_ARG_PTR);
6125      6241      END;
6126      6242
6127      6243
6128      6244      ! Announce the step.
6129      6245      ANNOUNCE_EVENT (.EVENT, .EXCEPTION_TYPE);
6130      6246
6131      6247
6132      6248      ! Delete the step.
6133      6249
6134      6250      EVENT [EVENT$V_DELETED] = TRUE;
6135      6251
6136      6252
6137      6253      ! DEBUG will stop for commands.
6138      6254
6139      6255      EVENT$B_STOP_USER = TRUE;
```



```

: 6140      6256      END;
: 6141      6257
: 6142      6258
: 6143      6259      ! Return - we haven't executed any new commands....
: 6144      6260
: 6145      6261      RETURN FALSE;
: 6146      6262      END;
: 6147      6263
: 6148      6264
: 6149      6265      ! This is not a steps (or skips) entry. Process normally.
: 6150      6266      ! Update the /AFTER count. If it becomes equal to 0, then the entry is
: 6151      6267      ! now active.
: 6152      6268
: 6153      6269      IF ((EVENT [EVENT$AFTER_COUNT] =
: 6154      6270      .EVENT [EVENT$AFTER_COUNT] - 1
: 6155      6271      ) EQLU 0
: 6156      6272      )
: 6157      6273      THEN
: 6158      6274      BEGIN
: 6159      6275
: 6160      6276      ! We've passed the /AFTER test. Now set the /AFTER count to 1.
: 6161      6277      !
: 6162      6278      EVENT [EVENT$AFTER_COUNT] = 1;
: 6163      6279
: 6164      6280
: 6165      6281      ! If there is a WHEN expression, process it....
: 6166      6282      !
: 6167      6283      IF .EVENT [EVENT$WHEN] NEQA 0
: 6168      6284      THEN
: 6169      6285      BEGIN
: 6170      6286      PARSE_WHEN_CONDITION(.EVENT);
: 6171      6287
: 6172      6288
: 6173      6289
: 6174      6290      ! If the condition is false, return FALSE.
: 6175      6291      !
: 6176      6292      IF NOT .WHEN_CONDITION
: 6177      6293      THEN
: 6178      6294      RETURN FALSE;
: 6179      6295      END;
: 6180      6296
: 6181      6297
: 6182      6298      ! If this event is temporary, flag it as deleted now. This shouldn't
: 6183      6299      ! affect any further processing, but the entry will be deleted when
: 6184      6300      ! we go off to the user program.
: 6185      6301
: 6186      6302      IF .EVENT [EVENT$V_ONCE_ONLY]
: 6187      6303      THEN
: 6188      6304      EVENT [EVENT$V_DELETED] = TRUE;
: 6189      6305
: 6190      6306
: 6191      6307      ! Flag a stop, if this is a BREAK.
: 6192      6308      !
: 6193      6309      IF .EVENT [EVENT$B_CMD_TYPE] EQLU EVENT$K_TYPE_BREAK OR
: 6194      6310      .EVENT [EVENT$B_CMD_TYPE] EQLU EVENT$K_TYPE_WATCH
: 6195      6311      THEN
: 6196      6312      EVENT$B_STOP_USER = TRUE;
```



```
6197 6313 3
6198 6314 3
6199 6315 3
6200 6316 3
6201 6317 3
6202 6318 3
6203 6319 3
6204 6320 3
6205 6321 3
6206 6322 3
6207 6323 3
6208 6324 3
6209 6325 3
6210 6326 3
6211 6327 4
6212 6328 4
6213 6329 4
6214 6330 4
6215 6331 4
6216 6332 4
6217 6333 4
6218 6334 4
6219 6335 4
6220 6336 4
6221 6337 4
6222 6338 4
6223 6339 4
6224 6340 4
6225 6341 4
6226 6342 4
6227 6343 3
6228 6344 3
6229 6345 3
6230 6346 3
6231 6347 3
6232 6348 3
6233 6349 3
6234 6350 3
6235 6351 4
6236 6352 4
6237 6353 4
6238 6354 4
6239 6355 4
6240 6356 4
6241 6357 4
6242 6358 4
6243 6359 4
6244 6360 4
6245 6361 4
6246 6362 4
6247 6363 4
6248 6364 4
6249 6365 4
6250 6366 4
6251 6367 4
6252 6368 4
6253 6369 4
```

```
! Case on the kind of event entry, to determine how to process it.
```

```
CASE .EVENT [EVENT$B_CMD_KIND]
FROM EVENT$K_KIND_ACC TO EVENT$K_KIND_EXC OF
  SET
    [EVENT$K_KIND_ACC]:
      ANNOUNCE_EVENT (.EVENT, .EXCEPTION_TYPE);
    [EVENT$K_KIND_INS]:
      ANNOUNCE_EVENT (.EVENT, .EXCEPTION_TYPE);
    [EVENT$K_KIND_EXC]:
      BEGIN
```

```
! This is an exception break or trace, so flag it.
```

```
DBG$GB_EXC_BRE_FLAG = TRUE;
```

```
! Format and output the message.
```

```
DBG$PUTMSG (.SIGNAL_ARG_PTR);
```

```
! Announce the break.
```

```
ANNOUNCE_EVENT (.EVENT, .EXCEPTION_TYPE);
END;
```

```
TES;
```

```
! If there is a DO associated with this entry, do it.
```

```
IF .EVENT [EVENT$L_DO] NEQA 0
THEN
  BEGIN
```

```
! We need a local pointer to the DO command string.
```

```
LOCAL
  DO_LENGTH,
  DO_STRING : REF VECTOR [, BYTE];
```

```
! Point to the DO LIST entry.
```

```
DO_LIST_ENTRY = .EVENT [EVENT$L_DO];
```

```
! Point to the command string.
```

```
DO_LENGTH = .((DO_LIST_ENTRY[EVENT$L_DO_LIST_POINT])<0,16,0>);
```



```
: 6254      6370      4
: 6255      6371      4
: 6256      6372      4
: 6257      6373      4
: 6258      6374      4
: 6259      6375      4
: 6260      6376      4
: 6261      6377      4
: 6262      6378      4
: 6263      6379      4
: 6264      6380      4
: 6265      6381      4
: 6266      6382      4
: 6267      6383      4
: 6268      6384      4
: 6269      6385      4
: 6270      6386      4
: 6271      6387      4
: 6272      6388      4
: 6273      6389      4
: 6274      6390      4
: 6275      6391      4
: 6276      6392      4
: 6277      6393      4
: 6278      6394      4
: 6279      6395      4
: 6280      6396      4
: 6281      6397      4
: 6282      6398      4
: 6283      6399      4
: 6284      6400      4
: 6285      6401      4
: 6286      6402      4
: 6287      6403      4
: 6288      6404      4
: 6289      6405      3
: 6290      6406      2
: 6291      6407      2
: 6292      6408      2
: 6293      6409      2
: 6294      6410      2
: 6295      6411      2
: 6296      6412      1
```

```
DO STRING = DBG$GET_MEMORY((.DO_LENGTH/%UPVAL)+1);
CH$MOVE(.DO_LENGTH,.DO_LIST_ENTRY[EVENT$%L_DO_LIST_POINT]+2,.DO_STRING);
DO_STRING[.DO_LENGTH] = 0;
```

```
! Add a link to the command input stream. First remember the
! current pointer to the head of the CIS stack, and the number
! of levels of nesting.
```

```
SAVED_CISHEAD = .DBG$GL_CISHEAD;
SAVED_LEVEL = .DBG$GL_CIS_LEVELS;
IF NOT DBG$NCIS_ADD (.DO_STRING, .DO_LENGTH, CIS_INPBUF,
0, 0, MESSAGE_VECT)
```

```
THEN
CALLG (.MESSAGE_VECT, LIB$SIGNAL);
```

```
! Process the DO clause. We repeatedly call COMMAND_PROC, each
! call executing one command from the DO list. We exit the
! loop if either: 1) the user said STEP or GO, thus resulting
! in the TAKE_CMD flag becoming false, or 2) We exhaust the
! commands in the current CIS, and get back to the SAVED_CISHEAD
! level.
```

```
DBG$NINITIALIZE ();
WHILE (.DBG$GB_TAKE_CMD) AND
(.DBG$GL_CISHEAD NEQ .SAVED_CISHEAD) AND
(.DBG$GL_CIS_LEVELS GTR .SAVED_LEVEL) DO
DBG$COMMAND_PROC ();
```

```
! If the user typed a GO, STEP, EXIT, or CONTINUE, return true,
! otherwise false.
```

```
RETURN NOT .DBG$GB_TAKE_CMD;
END;
```

```
END;
```

```
! Return FALSE since the /AFTER count hasn't run out.
```

```
RETURN FALSE;
```

```
END;
```

OFFC 00000

```
5B 00000000G 00 9E 00002
5A 00000000G 00 9E 00009
59 00000000' EF 9E 00010
58 00000000G 00 9E 00017
5E          04 C2 0001E
6D 012A CF DE 00021
54 08 AC D0 00026
```

```
.ENTRY DBG$PROCESS_EVENT, Save R2,R3,R4,R5,R6,R7,- : 6142
R8,R9,R10,RT1
MOVAB DBG$GL_CIS_LEVELS, R11
MOVAB DBG$GL_CISHEAD, R10
MOVAB EVENT$%STOP_USER, R9
MOVAB DBG$PUTMSG, R8
SUBL2 #4, SP
MOVAL 17$, (FP) : 6170
MOVL SIGNAL_ARG_PTR, R4 : 6191
```


	55	04	A4	D0	0002A	MOVL	4(R4), EXCEPTION_TYPE	:		
	50	04	AC	D0	0002E	MOVL	EVENT, R0	:	6197	
	04	14	A0	91	00032	CMPB	20(R0), #4	:		
			16	12	00036	BNEQ	2\$:		
17	AC	80	8F	88	00038	BISB2	#128, 23(R0)	:	6200	
	50	04	AC	D0	0003D	1\$: MOVL	EVENT, R0	:	6201	
	04	14	A0	91	00041	CMPB	20(R0), #4	:		
			07	12	00045	BNEQ	2\$:		
04	AC	08	A0	D0	00047	MOVL	8(R0), EVENT	:	6202	
			EF	11	0004C	BRB	1\$:		
	52	04	AC	D0	0004E	2\$: MOVL	EVENT, R2	:	6208	
	53	14	A2	9E	00052	MOVAB	20(R2), R3	:		
	03		63	91	00056	CMPB	(R3), #3	:		
			29	12	00059	BNEQ	5\$:		
		1C	A2	D7	0005B	DECL	28(R2)	:	6216	
			21	12	0005E	BNEQ	4\$:	6217	
	02	01	A3	91	00060	CMPB	1(R3), #2	:	6228	
			0C	12	00064	BNEQ	3\$:		
00000000G	00		01	90	00066	MOVB	#1, DBG\$GB_EXC_BRE_FLAG	:	6235	
			54	DD	0006D	PUSHL	R4	:	6240	
	68		01	FB	0006F	CALLS	#1, DBG\$PUTMSG	:		
			24	BB	00072	3\$: PUSHR	#^M<R2,R5>	:	6245	
0000V	CF		02	FB	00074	CALLS	#2, ANNOUNCE_EVENT	:		
03	A3	80	8F	88	00079	BISB2	#128, 3(R3)	:	6250	
	69		01	90	0007E	MOVB	#1, EVENT\$B_STOP_USER	:	6255	
			00C8	31	00081	4\$: BRW	16\$:	6261	
		1C	A2	D7	00084	5\$: DECL	28(R2)	:	6270	
			F8	12	00087	BNEQ	4\$:	6271	
1C	A2		01	D0	00089	MOVL	#1, 28(R2)	:	6279	
		20	A2	D5	0008D	TSTL	32(R2)	:	6284	
			0E	13	00090	BEQL	6\$:		
			52	DD	00092	PUSHL	R2	:	6287	
0000V	CF		01	FB	00094	CALLS	#1, PARSE_WHEN_CONDITION	:		
	E1	00000000'	EF	E9	00099	BLBC	WHEN_CONDITION, 4\$:	6292	
05	63		1A	E1	000A0	6\$: BBC	#26, (R3), 7\$:	6302	
	03	A3	80	8F	88	000A4	BISB2	#128, 3(R3)	:	6304
			63	95	000A9	7\$: TSTB	(R3)	:	6309	
			05	13	000AB	BEQL	8\$:		
	02		63	91	000AD	CMPB	(R3), #2	:	6310	
			03	12	000B0	BNEQ	9\$:		
	69		01	90	000B2	8\$: MOVB	#1, EVENT\$B_STOP_USER	:	6312	
02	00	01	A3	8F	000B5	9\$: CASEB	1(R3), #0, #2	:	6317	
0008	0014		0014		000BA	10\$: .WORD	12\$-10\$,-	:		
							12\$-10\$,-	:		
							11\$-10\$:		
			0C	11	000C0	BRB	12\$:	6324	
00000000G	00		01	90	000C2	11\$: MOVB	#1, DBG\$GB_EXC_BRE_FLAG	:	6332	
			54	DD	000C9	PUSHL	R4	:	6337	
	68		01	FB	000CB	CALLS	#1, DBG\$PUTMSG	:		
			24	BB	000CE	12\$: PUSHR	#^M<R2,R5>	:	6342	
0000V	CF		02	FB	000D0	CALLS	#2, ANNOUNCE_EVENT	:		
		24	A2	D5	000D5	TSTL	36(R2)	:	6349	
			72	13	000D8	BEQL	16\$:		
	50	24	A2	D0	000DA	MOVL	36(R2), DO_LIST_ENTRY	:	6363	
	52	0C	A0	D0	000DE	MOVL	12(DO_LIST_ENTRY), R2	:	6368	
	56		62	3C	000E2	MOVZWL	(R2), DO_LENGTH	:		
50	56		04	C7	000E5	DIVL3	#4, DO_LENGTH, R0	:	6370	

00000000G	00	01	A0	9F	000E9	PUSHAB	1(R0)	
	57		01	FB	000EC	CALLS	#1, DBG\$GET_MEMORY	
67	02		50	D0	000F3	MOVL	R0, DO_STRING	
	A2		56	28	000F6	MOV3	DO_LENGTH, 2(R2), (DO_STRING)	6371
			6647	94	000FB	CLRB	(DO_LENGTH)[DO_STRING]	6372
	53		6A	D0	000FE	MOVL	DBG\$GL_CISHEAD, SAVED_CISHEAD	6379
	52		6B	D0	00101	MOVL	DBG\$GL_CIS_LEVELS, SAVED_LEVEL	6380
			5E	DD	00104	PUSHL	SP	6381
			7E	7C	00106	CLRQ	-(SP)	
			02	DD	00108	PUSHL	#2	
			56	DD	0010A	PUSHL	DO_LENGTH	
			57	DD	0010C	PUSHL	DO_STRING	
00000000G	00		06	FB	0010E	CALLS	#6, DBG\$NCIS_ADD	
	08		50	E8	00115	BLBS	R0, 13\$	
00000000G	00	00	BE	FA	00118	CALLG	@MESSAGE_VECT, LIB\$SIGNAL	6384
00000000G	00		00	FB	00120	CALLS	#0, DBG\$NINITIALIZE	6394
	13	00000000G	00	E9	00127	BLBC	DBG\$GB_TAKE_CMD, 15\$	6395
	53		6A	D1	0012E	CMPL	DBG\$GL_CISHEAD, SAVED_CISHEAD	6396
			0E	13	00131	BEQL	15\$	
	52		6B	D1	00133	CMPL	DBG\$GL_CIS_LEVELS, SAVED_LEVEL	6397
			09	15	00136	BLEQ	15\$	
00000000G	00		00	FB	00138	CALLS	#0, DBG\$COMMAND_PROC	6398
			E6	11	0013F	BRB	14\$	
	50	00000000G	00	9A	00141	MOVZBL	DBG\$GB_TAKE_CMD, R0	6404
	50		50	D2	00148	MCOML	R0, R0	
				04	0014B	RET		
			50	D4	0014C	CLRL	R0	6412
			04	0014E	RET			
			0000	0014F	17\$:	.WORD	Save nothing	6170
			7E	D4	00151	CLRL	-(SP)	
			5E	DD	00153	PUSHL	SP	
	7E	04	AC	7D	00155	MOVQ	4(AP), -(SP)	
00000000G	00		03	FB	00159	CALLS	#3, DBG\$FINAL_HANDL	
			04	00160	RET			

; Routine Size: 353 bytes, Routine Base: DBG\$CODE + 2438


```

6298 6413 1 GLOBAL ROUTINE DBG$UPDATE_WATCHPOINTS: NOVALUE =
6299 6414 1
6300 6415 1 FUNCTION
6301 6416 1     This routine updates the values in the watch point events. This
6302 6417 1     is done in order to keep the watched values current even after a
6303 6418 1     DEPOSIT.
6304 6419 1
6305 6420 1 INPUTS
6306 6421 1     NONE
6307 6422 1
6308 6423 1 OUTPUTS
6309 6424 1     All watch point event entries are updated with their current value.
6310 6425 1
6311 6426 1
6312 6427 2 BEGIN
6313 6428 2
6314 6429 2 LOCAL
6315 6430 2     EVENT_ENTRY      : REF EVENT$EVENT_DESCRIPTOR,
6316 6431 2     LENGTH,
6317 6432 2     NEW_VALDESC      : REF DBG$VALDESC,
6318 6433 2     OLD_VALDESC      : REF DBG$VALDESC;
6319 6434 2
6320 6435 2
6321 6436 2     ! Starting at the head of the command queue, update each watch point
6322 6437 2     ! entry.
6323 6438 2
6324 6439 2     EVENT_ENTRY = .EVENT$CMD_QUEUE [L_QUEUE_FLINK];
6325 6440 2     WHILE .EVENT_ENTRY NEQA EVENT$CMD_QUEUE DO
6326 6441 3         BEGIN
6327 6442 3
6328 6443 3             ! If this is a watch point entry, update it.
6329 6444 3
6330 6445 3             IF .EVENT_ENTRY[EVENT$B_SUB_KIND] EQL EVENT$K_ACC_MDFY
6331 6446 3             THEN
6332 6447 4                 BEGIN
6333 6448 4
6334 6449 4                     ! Get the byte length of the entry's value. Note that we actually
6335 6450 4                     ! get it's bit length and round up.
6336 6451 4
6337 6452 4                     LENGTH = DBG$DATA_LENGTH(EVENT_ENTRY[EVENT$A_VMSDESC]);
6338 6453 4                     LENGTH = (.LENGTH + 7) ^ -3;
6339 6454 4
6340 6455 4                     ! Make a new value descriptor from the VMS descriptor in this
6341 6456 4                     ! event entry. This will then contain the new current value
6342 6457 4                     ! we are looking for.
6343 6458 4
6344 6459 4                     NEW_VALDESC = DBG$MAKE_VAL_DESC(EVENT_ENTRY[EVENT$A_VMSDESC],
6345 6460 4                                     DBG$K_VALUE_DESC);
6346 6461 4
6347 6462 4                     ! Now copy the "new" value into the old value descriptor that
6348 6463 4                     ! hangs of this event entry.
6349 6464 4
6350 6465 4                     OLD_VALDESC = .EVENT_ENTRY[EVENT$L_VALDESC];
6351 6466 4                     CH$MOVE(.LENGTH,
6352 6467 4                         NEW_VALDESC [DBG$A_VALUE_ADDRESS],
6353 6468 4                         OLD_VALDESC [DBG$A_VALUE_ADDRESS]);
6354 6469 3                 END;
```



```

: 6355      6470      3
: 6356      6471      3
: 6357      6472      3
: 6358      6473      3
: 6359      6474      2
: 6360      6475      1

```

! Point to the next Command Queue entries.
EVENT_ENTRY = .EVENT_ENTRY [EVENT\$L_CMD_FLINK];
END;
END;

				07FC 00000	.ENTRY	DBG\$UPDATE WATCHPOINTS, Save R2,R3,R4,R5,-	: 6413
		5A 00000000'	EF 9E 00002		MOVAB	R6,R7,R8,R9,R10	
		56	6A D0 00009		MOVL	EVENT\$CMD_QUEUE, R10	: 6439
		50	6A 9E 0000C	1\$:	MOVAB	EVENT\$CMD_QUEUE, EVENT_ENTRY	: 6440
		50	56 D1 0000F		CMPL	EVENT_ENTRY, R0	
			3C 13 00012		BEQL	3\$	
		02	A6 91 00014		CMPB	22(EVENT_ENTRY), #2	: 6445
			31 12 00018		BNEQ	2\$	
			A6 9F 0001A		PUSHAB	52(EVENT_ENTRY)	: 6452
		00000000G	01 FB 0001D		CALLS	#1, DBG\$DATA_LENGTH	
		59	50 D0 00024		MOVL	R0, LENGTH	
		50	A9 9E 00027		MOVAB	7(R9), R0	: 6453
59		50	FD 8F 78 0002B		ASHL	#-3, R0, LENGTH	
		7E	7A 8F 9A 00030		MOVZBL	#122, -(SP)	: 6459
			A6 9F 00034		PUSHAB	52(EVENT_ENTRY)	
		00000000G	02 FB 00037		CALLS	#2, DBG\$MAKE_VAL_DESC	
		58	50 D0 0003E		MOVL	R0, NEW_VALDESC	
		57	A6 D0 00041		MOVL	48(EVENT_ENTRY), OLD_VALDESC	: 6465
20	A7	20	59 28 00045		MOVC3	LENGTH, 32(NEW_VALDESC), 32(OLD_VALDESC)	: 6468
		56	66 D0 0004B	2\$:	MOVL	(EVENT_ENTRY), EVENT_ENTRY	: 6473
			BC 11 0004E		BRB	1\$: 6440
			04 00050	3\$:	RET		: 6475

; Routine Size: 81 bytes, Routine Base: DBG\$CODE + 2599

```

6362 6476 1 ROUTINE ANNOUNCE_EVENT(EVENT_ENTRY, EXCEPTION_TYPE): NOVALUE =
6363 6477 1
6364 6478 1 FUNCTION
6365 6479 1     This routine announces an event entry if the /SILENT flag is false.
6366 6480 1
6367 6481 1 INPUTS
6368 6482 1     EVENT_ENTRY :           Pointer to event entry.
6369 6483 1
6370 6484 1 OUTPUTS
6371 6485 1     Announces the event entry via DBG$PRINT.
6372 6486 1
6373 6487 1
6374 6488 1 BEGIN
6375 6489 2
6376 6490 2 MAP
6377 6491 2     EVENT_ENTRY :       REF EVENT$EVENT_DESCRIPTOR, ! Event Entry Pointer
6378 6492 2     EXCEPTION_TYPE :   BLOCK [%UPVAL, BYTE];
6379 6493 2
6380 6494 2
6381 6495 2 LOCAL
6382 6496 2     VMS_DESC: DBG$STG_DESC,
6383 6497 2     VAL_DESC: REF DBG$VALDESC,
6384 6498 2     STRING_DESCRIPTOR : BLOCK [8, BYTE],
6385 6499 2     OUTPUT_BUFFER : VECTOR [TTY_OUT_WIDTH, BYTE],
6386 6500 2     DUMMY,
6387 6501 2
6388 6502 2     ADDRESS:                ! Entry address
6389 6503 2
6390 6504 2
6391 6505 2 ! If the /SILENT flag is set, return without announcing.
6392 6506 2 ! (This can be set by SET BREAK/SILENT or STEP/SILENT).
6393 6507 2
6394 6508 2 IF .EVENT_ENTRY[EVENT$V_SILENT] THEN RETURN;
6395 6509 2
6396 6510 2
6397 6511 2 ! Purge type ahead.
6398 6512 2
6399 6513 2 DBG$GL_OUTPRAB [RAB$V_CCO] = TRUE;
6400 6514 2
6401 6515 2
6402 6516 2 ! Setup a string descriptor.
6403 6517 2
6404 6518 2 STRING_DESCRIPTOR [DSC$B_DTYPE] = DSC$K_DTYPE_T;
6405 6519 2 STRING_DESCRIPTOR [DSC$B_CLASS] = DSC$K_CLASS_S;
6406 6520 2 STRING_DESCRIPTOR [DSC$A_POINTER] = 0;
6407 6521 2 STRING_DESCRIPTOR [DSC$W_LENGTH] = 0;
6408 6522 2
6409 6523 2
6410 6524 2 ! Print the event type (break, trace, watch or step to - point).
6411 6525 2
6412 6526 2 DBG$PRINT ($AC ('!AC'),
6413 6527 2     SELECT ONE .EVENT_ENTRY [EVENT$B_CMD_TYPE] OF
6414 6528 2     SET
6415 6529 2     [EVENT$K_TYPE_BREAK]:
6416 6530 2         $AC ('break');
6417 6531 2     [EVENT$K_TYPE_TRACE]:
6418 6532 2         $AC ('trace');
```



```
.. 6419      6533      2
.. 6420      6534      2
.. 6421      6535      2
.. 6422      6536      2
.. 6423      6537      2
.. 6424      6538      2
.. 6425      6539      2
.. 6426      6540      2
.. 6427      6541      2
.. 6428      6542      2
.. 6429      6543      2
.. 6430      6544      2
.. 6431      6545      2
.. 6432      6546      2
.. 6433      6547      2
.. 6434      6548      2
.. 6435      6549      2
.. 6436      6550      2
.. 6437      6551      2
.. 6438      6552      2
.. 6439      6553      2
.. 6440      6554      2
.. 6441      6555      2
.. 6442      6556      2
.. 6443      6557      2
.. 6444      6558      2
.. 6445      6559      4
.. 6446      6560      4
.. 6447      6561      4
.. 6448      6562      4
.. 6449      6563      4
.. 6450      6564      4
.. 6451      6565      4
.. 6452      6566      4
.. 6453      6567      4
.. 6454      6568      4
.. 6455      6569      4
.. 6456      6570      4
.. 6457      6571      4
.. 6458      6572      4
.. 6459      6573      4
.. 6460      6574      4
.. 6461      6575      4
.. 6462      6576      4
.. 6463      6577      4
.. 6464      6578      4
.. 6465      6579      4
.. 6466      6580      4
.. 6467      6581      4
.. 6468      6582      4
.. 6469      6583      4
.. 6470      6584      4
.. 6471      6585      4
.. 6472      6586      4
.. 6473      6587      4
.. 6474      6588      5
.. 6475      6589      5
```

```
      [EVENT$K_TYPE_WATCH]:
      SAC ('watch');
      [EVENT$K_TYPE_STEPS]:
      SAC ('stepped');
      TES
    );

! Display the event kind (access, exception, instruction).
! SELECT ONE .EVENT_ENTRY [EVENT$B_CMD_KIND] OF
  SET

! This is an access (read, write, modify, execute) kind
! of event point. Display the address.
[EVENT$K_KIND_ACC]:
  BEGIN

! If this is not an /EXECUTE type (we're done with those), continue
! processing....
  IF .EVENT_ENTRY [EVENT$B_SUB_KIND] NEQU EVENT$K_ACC_EXEC
  THEN
    BEGIN

! Print the access kind if read, write, modify, or return.
! If the address is an entry point, indicate that it is a
! routine.
    DBG$PRINT ($AC (' !AC!AC'),
      IF .EVENT_ENTRY [EVENT$B_CMD_TYPE] NEQU
      EVENT$K_TYPE_WATCH
      THEN
        SELECT ONE .EVENT_ENTRY [EVENT$B_SUB_KIND] OF
          SET
            [EVENT$K_ACC_READ]:
              SAC ('on read of ');
            [EVENT$K_ACC_WRIT]:
              SAC ('on write of ');
            [EVENT$K_ACC_MDFY]:
              SAC ('on modify of ');
            [EVENT$K_ACC_RTRN]:
              SAC ('on return from ');
          TES
        ELSE
          SAC ('of '),
        IF .EVENT_ENTRY [EVENT$V_BREAK_ADDRESS]
        THEN
          SAC ('routine ')
        ELSE
          BEGIN
            IF DBG$IS_IT_ENTRY
```

```
: 6476      6590      S
: 6477      6591
: 6478      6592
: 6479      6593
: 6480      6594
: 6481      6595
: 6482      6596
: 6483      6597
: 6484      6598
: 6485      6599
: 6486      6600
: 6487      6601
: 6488      6602
: 6489      6603
: 6490      6604
: 6491      6605
: 6492      6606
: 6493      6607
: 6494      6608
: 6495      6609
: 6496      6610
: 6497      6611
: 6498      6612
: 6499      6613
: 6500      6614
: 6501      6615
: 6502      6616
: 6503      6617
: 6504      6618
: 6505      6619
: 6506      6620
: 6507      6621
: 6508      6622
: 6509      6623
: 6510      6624
: 6511      6625
: 6512      6626
: 6513      6627
: 6514      6628
: 6515      6629
: 6516      6630
: 6517      6631
: 6518      6632
: 6519      6633
: 6520      6634
: 6521      6635
: 6522      6636
: 6523      6637
: 6524      6638
: 6525      6639
: 6526      6640
: 6527      6641
: 6528      6642
: 6529      6643
: 6530      6644
: 6531      6645
: 6532      6646      S
```

```
      (.EVENT_ENTRY [EVENT$$_ADDRESS] - 2)
    THEN
      $AC ('routine ')
    ELSE
      $AC ('')
    END
  );

! Print the address.
! IF .EVENT_ENTRY [EVENT$$_CMD_TYPE] EQLU EVENT$$_TYPE_STEPS
! THEN
!   BEGIN
!     LOCAL
!       FROM_ADDR;

!   ! Display 'ROUTINE' if the saved 'from' address points
!   ! after an entry point.
!   FROM_ADDR = .EVENT_ENTRY [EVENT$$_USERS_PC];
!   IF DBG$$_IS_IT_ENTRY- (.FROM_ADDR - 2)
!   THEN
!     BEGIN
!       DBG$$_PRINT ($AC ('routine '));
!       FROM_ADDR = .FROM_ADDR - 2;
!     END;

!   ! Print the address, but symbolize it first.
!   DBG$$_PRINT_IDENTIFIER_PC (.FROM_ADDR);
!   END
! ELSE
!   DBG$$_PRINT_IDENTIFIER (.EVENT_ENTRY [EVENT$$_PRIMARY]);
! END;

! Depending on the command type....
! IF .EVENT_ENTRY [EVENT$$_CMD_TYPE] EQLU EVENT$$_TYPE_STEPS
! THEN
!   DBG$$_PRINT ($AC (' to '))
! ELSE
!   DBG$$_PRINT ($AC (' at '));
! END;

! This is an instruction type.
! [EVENT$$_KIND_INS]:
! BEGIN
```



```

: 6533      6647      3
: 6534      6648
: 6535      6649
: 6536      6650
: 6537      6651
: 6538      6652
: 6539      6653
: 6540      6654
: 6541      6655
: 6542      6656
: 6543      6657
: 6544      6658
: 6545      6659
: 6546      6660
: 6547      6661
: 6548      6662
: 6549      6663
: 6550      6664
: 6551      6665
: 6552      6666
: 6553      6667
: 6554      6668
: 6555      6669
: 6556      6670
: 6557      6671
: 6558      6672
: 6559      6673
: 6560      6674
: 6561      6675
: 6562      6676
: 6563      6677
: 6564      6678
: 6565      6679
: 6566      6680
: 6567      6681
: 6568      6682
: 6569      6683
: 6570      6684
: 6571      6685
: 6572      6686
: 6573      6687
: 6574      6688
: 6575      6689
: 6576      6690
: 6577      6691
: 6578      6692
: 6579      6693
: 6580      6694
: 6581      6695
: 6582      6696
: 6583      6697
: 6584      6698
: 6585      6699
: 6586      6700
: 6587      6701
: 6588      6702
: 6589      6703      4

! Display the instruction type (calls, branches, all, or user
! list.
IF .EVENT_ENTRY [EVENT$B_CMD_TYPE] NEQU EVENT$K_TYPE_STEPS
THEN
  DBG$PRINT
    ($AC (' on !AC'),
    SELECTONE .EVENT_ENTRY [EVENT$B_SUB_KIND] OF
      SET
        [EVENT$K_INS_CALL]:
          SAC ('calls');
        [EVENT$K_INS_BRAN]:
          SAC ('branches');
        [EVENT$K_INS_EVR]:
          SAC ('instruction');
        [EVENT$K_INS_USER]:
          SAC ('instruction(s)');
        [EVENT$K_INS_LINE]:
          SAC ('lines');
      TES
    );

! Depending on the command type....
IF .EVENT_ENTRY [EVENT$B_CMD_TYPE] EQLU EVENT$K_TYPE_STEPS
THEN
  DBG$PRINT ($AC (' to '))
ELSE
  DBG$PRINT ($AC (' at '));
END;

! This is an exception type.
[EVENT$K_KIND_EXC]:
  BEGIN
    LOCAL
      DST_PTR : REF DST$RECORD,
      RST_PTR : REF RST$ENTRY,
      TRAP_OR_FAULT,
      LINE_NO,
      STMT_NO,
      STATUS;

! Determine the nature (trap or fault) of the exception.
IF .EXCEPTION_TYPE [ST$V_FAC_NO] EQL 0
THEN
  BEGIN
    SELECTONE .EXCEPTION_TYPE OF
      SET
        [SS$_ASTFLT, SS$_BREAK, SS$_COMPAT,
```

```
: 6590      6704  4      SSS_OPCCUS, SSS_OPDEC, SSS_RADRMOD, SSS_ROPRAND]:
: 6591      6705  4      TRAP_OR_FAULT = FAULT_EXC;
: 6592      6706  4
: 6593      6707  4      [OTHERWISE]:
: 6594      6708  4      TRAP_OR_FAULT = TRAP_EXC;
: 6595      6709  4
: 6596      6710  4      TES;
: 6597      6711  4
: 6598      6712  4      END
: 6599      6713  4
: 6600      6714  4
: 6601      6715  4      ! If the facility is not SSS, then we have a trap, meaning that the
: 6602      6716  4      ! PC points to the instruction after the LIBSSIGNAL or LIBSSTOP call
: 6603      6717  4      ! that raised the exception. (Non-SSS exceptions can only be sig-
: 6604      6718  4      ! nalled, not raised by the hardware.)
: 6605      6719  4
: 6606      6720  3      ELSE
: 6607      6721  3      TRAP_OR_FAULT = TRAP_EXC;
: 6608      6722  3
: 6609      6723  3
: 6610      6724  3      ! The PC is the address to display.
: 6611      6725  3      !
: 6612      6726  3      ADDRESS = .DBG$RUNFRAME [DBG$L_USER_PC];
: 6613      6727  3
: 6614      6728  3
: 6615      6729  3      ! Display the appropriate message.
: 6616      6730  3      !
: 6617      6731  3      DBG$PRINT ($AC (' on exception '));
: 6618      6732  3      IF .TRAP_OR_FAULT EQL TRAP_EXC
: 6619      6733  3      THEN
: 6620      6734  3      DBG$PRINT ($AC ('preceding '))
: 6621      6735  3
: 6622      6736  3      ELSE
: 6623      6737  3      DBG$PRINT ($AC ('at '));
: 6624      6738  3
: 6625      6739  2      END;
: 6626      6740  2
: 6627      6741  2      TES;
: 6628      6742  2
: 6629      6743  2
: 6630      6744  2      ! If this is a /MODIFY event (we used ACCVIO's), the address of the
: 6631      6745  2      ! offending instruction is in ACCVIO_PC, otherwise it's just the
: 6632      6746  2      ! current PC.
: 6633      6747  2
: 6634      6748  2      IF (.EVENT_ENTRY [EVENT$B_CMD_KIND] EQLU EVENT$K_KIND_ACC) AND
: 6635      6749  3      (.EVENT_ENTRY [EVENT$B_SUB_KIND] EQLU EVENT$K_ACC_MDFY)
: 6636      6750  2      THEN
: 6637      6751  2      ADDRESS = .ACCVIO_PC
: 6638      6752  2
: 6639      6753  2      ELSE
: 6640      6754  2      ADDRESS = .DBG$RUNFRAME [DBG$L_USER_PC];
: 6641      6755  2
: 6642      6756  2
: 6643      6757  2      ! Symbolize and print address (and instruction)
: 6644      6758  2      !
: 6645      6759  2      PRINT PC AND INSTRUCTION (.ADDRESS, .EVENT_ENTRY);
: 6646      6760  2      DBG$NEWLINE 7);
```



```

: 6647      6761      2
: 6648      6762      2
: 6649      6763      2
: 6650      6764      2
: 6651      6765      2
: 6652      6766      2
: 6653      6767      2
: 6654      6768      2
: 6655      6769      2
: 6656      6770      2
: 6657      6771      2
: 6658      6772      2
: 6659      6773      3
: 6660      6774      4
: 6661      6775      3
: 6662      6776      4
: 6663      6777      4
: 6664      6778      3
: 6665      6779      3
: 6666      6780      3
: 6667      6781      3
: 6668      6782      3
: 6669      6783      3
: 6670      6784      3
: 6671      6785      3
: 6672      6786      3
: 6673      6787      3
: 6674      6788      3
: 6675      6789      3
: 6676      6790      3
: 6677      6791      3
: 6678      6792      3
: 6679      6793      3
: 6680      6794      3
: 6681      6795      3
: 6682      6796      3
: 6683      6797      3
: 6684      6798      3
: 6685      6799      3
: 6686      6800      3
: 6687      6801      3
: 6688      6802      3
: 6689      6803      3
: 6690      6804      3
: 6691      6805      3
: 6692      6806      3
: 6693      6807      3
: 6694      6808      3
: 6695      6809      3
: 6696      6810      2
: 6697      6811      2
: 6698      6812      2
: 6699      6813      2
: 6700      6814      4
: 6701      6815      3
: 6702      6816      4
: 6703      6817      3

```

```

! If this is a modify-type, display the old and new values, and the
! BREAK message.
IF .EVENT_ENTRY [EVENT$B_SUB_KIND] EQLU EVENT$K_ACC_MDFY
THEN
  BEGIN

    ! Print the source line where the actual variable-access occurred if
    ! appropriate.
    IF (.EVENT_ENTRY[EVENT$V_OVRD_SOURCE] AND
        .EVENT_ENTRY[EVENT$V_STEP_SOURCE]) OR
        ((NOT .EVENT_ENTRY[EVENT$V_OVRD_SOURCE]) AND
         .DBG$GB_STP_PTR[EVENT$V_STEPPING_SOURCE])
    THEN
      PRINT_SOURCE(.ADDRESS);

    ! Print the old value and the new value of the watched location.
    DBG$PRINT($AC('  old value: '));
    DBG$PRINT_VALUE(.OLDVALUE, .DBG$GB_RADIX[DBG$B_RADIX_OUTPUT_OVER],
                    .DBG$GC_SIGN_FLAG, FALSE);
    DBG$NEWLINE();
    DBG$PRINT($AC('  new value: '));
    DBG$PRINT_VALUE(.NEWVALUE, .DBG$GB_RADIX[DBG$B_RADIX_OUTPUT_OVER],
                    .DBG$GC_SIGN_FLAG, FALSE);
    DBG$NEWLINE();

    ! Display the 'BREAK AT <address>' message.
    DBG$PRINT (IF .EVENT_ENTRY [EVENT$B_CMD_TYPE] NEQU
                EVENT$K_TYPE_TRACE
                THEN
                  $AC ('break at ')
                ELSE
                  $AC ('trace at ')
                );

    ! Symbolize and print address (and instruction)
    PRINT_PC_AND_INSTRUCTION (.DBG$RUNFRAME [DBG$L_USER_PC], .EVENT_ENTRY);
    DBG$NEWLINE ();
    END;

    ! Display the source line, if source stepping is indicated.
    IF .TYPE_SOURCE AND
        ((.EVENT_ENTRY[EVENT$V_OVRD_SOURCE] AND
         .EVENT_ENTRY[EVENT$V_STEP_SOURCE]) OR
         ((NOT .EVENT_ENTRY[EVENT$V_OVRD_SOURCE]) AND
          .DBG$GB_STP_PTR[EVENT$V_STEPPING_SOURCE]))
    THEN

```

```
: 6704      6818  2
: 6705      6819
: 6706      6820
: 6707      6821
: 6708      6822
: 6709      6823
: 6710      6824
: 6711      6825
: 6712      6826
: 6713      6827
: 6714      6828
: 6715      6829  1

THEN
  BEGIN
    PRINT_SOURCE DBGSRUNFRAME [DBG$USER_PC]);
    TYPE_SOURCE  FALSE;
  END;

! We are all done--return.
!
RETURN;

END;
```

```
.PSECT DBGSPLIT,NOWRT, SHR, PIC,0

        6D  20  20  20  66  6F  20  65  74  69  72  77  20  6E  6F  0B  0029F P.ACN: .ASCII <11>\on read of \
        6D  6F  72  66  20  6E  72  75  74  65  72  20  6E  6F  0C  002AB P.ACO: .ASCII <12>\on write of \
        6D  6F  72  66  20  6E  72  75  74  65  72  20  6E  6F  0D  002B8 P.ACP: .ASCII <13>\on modify of \
        6D  6F  72  66  20  6E  72  75  74  65  72  20  6E  6F  0E  002C6 P.ACQ: .ASCII <15>\on return from \
        6D  6F  72  66  20  6E  72  75  74  65  72  20  6E  6F  0F  002D5
        6D  6F  72  66  20  6E  72  75  74  65  72  20  6E  6F  03  002D6 P.ACR: .ASCII <3>\of \
        6D  6F  72  66  20  6E  72  75  74  65  72  20  6E  6F  08  002DA P.ACS: .ASCII <8>\routine \
        6D  6F  72  66  20  6E  72  75  74  65  72  20  6E  6F  08  002E3 P.ACT: .ASCII <8>\routine \
        6D  6F  72  66  20  6E  72  75  74  65  72  20  6E  6F  00  002EC P.ACU: .ASCII <0>
        6D  6F  72  66  20  6E  72  75  74  65  72  20  6E  6F  08  002ED P.ACV: .ASCII <8>\routine \
        6D  6F  72  66  20  6E  72  75  74  65  72  20  6E  6F  04  002F6 P.ACW: .ASCII <4>\ to \
        6D  6F  72  66  20  6E  72  75  74  65  72  20  6E  6F  04  002FB P.ACX: .ASCII <4>\ at \
        6D  6F  72  66  20  6E  72  75  74  65  72  20  6E  6F  07  00300 P.ACY: .ASCII <7>\ on !AC\
        6D  6F  72  66  20  6E  72  75  74  65  72  20  6E  6F  05  00308 P.ACZ: .ASCII <5>\calls\
        6D  6F  72  66  20  6E  72  75  74  65  72  20  6E  6F  08  0030E P.ADA: .ASCII <8>\branches\
        6D  6F  72  66  20  6E  72  75  74  65  72  20  6E  6F  0B  00317 P.ADB: .ASCII <11>\instruction\
        6D  6F  72  66  20  6E  72  75  74  65  72  20  6E  6F  0E  00323 P.ADC: .ASCII <14>\instruction(s)\
        6D  6F  72  66  20  6E  72  75  74  65  72  20  6E  6F  05  00332 P.ADD: .ASCII <5>\lines\
        6D  6F  72  66  20  6E  72  75  74  65  72  20  6E  6F  04  00338 P.ADE: .ASCII <4>\ to \
        6D  6F  72  66  20  6E  72  75  74  65  72  20  6E  6F  04  0033D P.ADF: .ASCII <4>\ at \
        6D  6F  72  66  20  6E  72  75  74  65  72  20  6E  6F  0E  00342 P.ADG: .ASCII <14>\ on exception \
        6D  6F  72  66  20  6E  72  75  74  65  72  20  6E  6F  0A  00351 P.ADH: .ASCII <10>\preceding \
        6D  6F  72  66  20  6E  72  75  74  65  72  20  6E  6F  03  0035C P.ADI: .ASCII <3>\at \
        6D  6F  72  66  20  6E  72  75  74  65  72  20  6E  6F  0E  00360 P.ADJ: .ASCII <14>\ old value: \
        6D  6F  72  66  20  6E  72  75  74  65  72  20  6E  6F  0E  0036F P.ADK: .ASCII <14>\ new value: \
        6D  6F  72  66  20  6E  72  75  74  65  72  20  6E  6F  09  0037E P.ADL: .ASCII <9>\break at \
        6D  6F  72  66  20  6E  72  75  74  65  72  20  6E  6F  09  00388 P.ADM: .ASCII <9>\trace at \
```

```
.PSECT DBGS$CODE,NOWRT, SHR, PIC,0
```

```
OFFC 00000 ANNOUNCE_EVENT:
```

```
.WORD Save R2,R3,R4,R5,R6,R7,R8,R9,R10,R11
```

```
: 6476
```


5B	00000000G	00	9E	00002	MOVAB	DBG\$IS_IT_ENTRY, R11	
5A	00000000G	00	9E	00009	MOVAB	DBG\$NEWLINE, R10	
59	00000000G	00	9E	00010	MOVAB	DBG\$RUNFRAME+64, R9	
58	000000000	EF	9E	00017	MOVAB	TYPE_SOURCE, R8	
57	00000000G	00	9E	0001E	MOVAB	DBG\$PRINT, R7	
56	000000000	EF	9E	00025	MOVAB	P.ACI, R6	
5E	FF68	CE	9E	0002C	MOVAB	-152(SP), SP	
52	04	AC	D0	00031	MOVL	EVENT_ENTRY, R2	6508
54	14	A	9E	00035	MOVAB	20(R2), R4	
01	03	A4	E9	00039	BLBC	3(R4), 1\$	
			04	0003D	RET		
00000000G	00	8F	88	0003E	BISB2	#128, DBG\$GL_OUTPRAB+7	6513
EC	AD	8F	D0	00046	MOVL	#17694720, STRING_DESCRIPTOR	6521
		AD	D4	0004E	CLRL	STRING_DESCRIPTOR+4	6520
		64	95	00051	TSTB	(R4)	6529
		05	12	00053	BNEQ	2\$	
50		66	9E	00055	MOVAB	P.ACI, R0	6530
		24	11	00058	BRB	6\$	
01		64	91	0005A	CMPB	(R4), #1	6531
		06	12	0005D	BNEQ	3\$	
50	06	A6	9E	0005F	MOVAB	P.ACJ, R0	6532
		19	11	00063	BRB	6\$	
02		64	91	00065	CMPB	(R4), #2	6533
		06	12	00068	BNEQ	4\$	
50	0C	A6	9E	0006A	MOVAB	P.ACK, R0	6534
		0E	11	0006E	BRB	6\$	
03		64	91	00070	CMPB	(R4), #3	6535
		05	13	00073	BEQL	5\$	
7E		01	CE	00075	MNEGL	#1, -(SP)	
		06	11	00078	BRB	7\$	
50	12	A6	9E	0007A	MOVAB	P.ACL, R0	6536
		50	DD	0007E	PUSHL	R0	
	FC	A6	9F	00080	PUSHAB	P.ACH	6526
67		02	FB	00083	CALLS	#2, DBG\$PRINT	
	01	A4	95	00086	TSTB	1(R4)	6550
		03	13	00089	BEQL	8\$	
		00AF	31	0008B	BRW	23\$	
03	02	A4	91	0008E	CMPB	2(R4), #3	6557
		03	12	00092	BNEQ	9\$	
		0097	31	00094	BRW	21\$	
06	19	A2	05	E1	BBC	#5, 25(R2), 10\$	6584
		50	A6	9E	MOVAB	P.ACS, R0	6586
			15	11	BRB	12\$	
7E	2C	A2	02	C3	SUBL3	#2, 44(R2), -(SP)	6590
		6B	01	FB	CALLS	#1, DBG\$IS_IT_ENTRY	
		06	50	E9	BLBC	R0, 11\$	
		50	A6	9E	MOVAB	P.ACT, R0	6592
			04	11	BRB	12\$	
		50	A6	9E	MOVAB	P.ACU, R0	6594
			50	DD	PUSHL	R0	6588
		02	64	91	CMPB	(R4), #2	6567
			34	13	BEQL	17\$	
			02	A4	TSTB	2(R4)	6572
			06	12	BNEQ	13\$	
		50	A6	9E	MOVAB	P.ACN, R0	6573
			2D	11	BRB	18\$	
		01	02	A4	CMPB	2(R4), #1	6574

		06	12	000CD		BNEQ	14\$			
50	2E	A6	9E	000CF		MOVAB	P.ACO, R0		6575	
		21	11	000D3		BRB	18\$			
02	02	A4	91	000D5	14\$:	CMPB	2(R4), #2		6576	
		06	12	000D9		BNEQ	15\$			
50	3B	A6	9E	000DB		MOVAB	P.ACP, R0		6577	
		15	11	000DF		BRB	18\$			
04	02	A4	91	000E1	15\$:	CMPB	2(R4), #4		6578	
		05	13	000E5		BEQL	16\$			
50		01	CE	000E7		MNEGL	#1, R0			
		0A	11	0C0EA		BRB	18\$			
50	49	A6	9E	000EC	16\$:	MOVAB	P.ACQ, R0		6579	
		04	11	000F0		BRB	18\$		6570	
50	59	A6	9E	000F2	17\$:	MOVAB	P.ACR, R0		6582	
		50	DD	000F6	18\$:	PUSHL	R0			
	1A	A6	9F	000F8		PUSHAB	P.ACM		6566	
67		03	FB	000FB		CALLS	#3, DBG\$PRINT			
03		64	91	000FE		CMPB	(R4), #3		6601	
		21	12	00101		BNEQ	20\$			
53	20	A2	D0	00103		MOVL	32(R2), FROM_ADDR		6611	
	FE	A3	9F	00107		PUSHAB	-2(FROM_ADDR)		6612	
6B		01	FB	0010A		CALLS	#1, DBG\$IS_IT_ENTRY			
09		50	E9	0010D		BLBC	R0, 19\$			
	70	A6	9F	00110		PUSHAB	P.ACV		6615	
67		01	FB	00113		CALLS	#1, DBG\$PRINT			
53		02	C2	00116		SUBL2	#2, FROM_ADDR		6616	
		53	DD	00119	19\$:	PUSHL	FROM_ADDR		6622	
00000000G	00	01	FB	0011B		CALLS	#1, DBG\$PRINT_IDENTIFIER_PC			
		0A	11	00122		BRB	21\$		6601	
	28	A2	DD	00124	20\$:	PUSHL	40(R2)		6626	
00000000G	00	01	FB	00127		CALLS	#1, DBG\$PRINT_IDENTIFIER			
	03	64	91	0C12E	21\$:	CMPB	(R4), #3		6633	
		05	12	00131		BNEQ	22\$			
	79	A6	9F	00133		PUSHAB	P.ACW		6635	
		6C	11	00136		BRB	33\$			
	7E	A6	9F	00138	22\$:	PUSHAB	P.ACX		6638	
		67	11	0013B		BRB	33\$			
01	01	A4	91	0013D	23\$:	CMPB	1(R4), #1		6645	
		63	12	00141		BNEQ	34\$			
03		64	91	00143		CMPB	(R4), #3		6652	
		4D	13	00146		BEQL	31\$			
05	02	A4	91	00148		CMPB	2(R4), #5		6658	
		07	12	0014C		BNEQ	24\$			
50	008B	C6	9E	0014E		MOVAB	P.ACZ, R0		6659	
		37	11	00153		BRB	29\$			
06	02	A4	91	00155	24\$:	CMPB	2(R4), #6		6660	
		07	12	00159		BNEQ	25\$			
50	0091	C6	9E	0015B		MOVAB	P.ADA, R0		6661	
		2A	11	00160		BRB	29\$			
08	02	A4	91	00162	25\$:	CMPB	2(R4), #8		6662	
		07	12	00166		BNEQ	26\$			
50	009A	C6	9E	00168		MOVAB	P.ADB, R0		6663	
		1D	11	0016D		BRB	29\$			
09	02	A4	91	0016F	26\$:	CMPB	2(R4), #9		6664	
		07	12	00173		BNEQ	27\$			
50	00A6	C6	9E	00175		MOVAB	P.ADC, R0		6665	
		10	11	0017A		BRB	29\$			

	07	02	A4	91	0017C	27\$:	CMPB	2(R4), #7	: 6666
			05	13	00180		BEQL	28\$	
	7E		01	CE	00182		MNEGL	#1, -(SP)	
			07	11	00185		BRB	30\$	
	50	00B5	C6	9E	00187	28\$:	MOVAB	P.ADD, R0	: 6667
			50	DD	0018C	29\$:	PUSHL	R0	
		0083	C6	9F	0018E	30\$:	PUSHAB	P.ACY	: 6655
	67		02	FB	00192		CALLS	#2, DBG\$PRINT	
	03		64	91	00195	31\$:	CMPB	(R4), #3	: 6674
			06	12	00198		BNEQ	32\$	
		00BB	C6	9F	0019A		PUSHAB	P.ADE	: 6676
			78	11	0019E		BRB	39\$	
		00C0	C6	9F	001A0	32\$:	PUSHAB	P.ADF	: 6678
			72	11	001A4	33\$:	BRB	39\$	
	02	01	A4	91	001A6	34\$:	CMPB	1(R4), #2	: 6684
			6F	12	001AA		BNEQ	40\$	
OFFF	8F	0A	AC	B3	001AC		BITW	EXCEPTION_TYPE+2, #4095	: 6697
			48	12	001B2		BNEQ	36\$	
	50	08	AC	D0	001B4		MOVL	EXCEPTION_TYPE, R0	: 6700
0000040C	8F		50	D1	001B8		CMPL	R0, #1036	: 6703
			36	13	001BF		BEQL	35\$	
00000414	8F		50	D1	001C1		CMPL	R0, #1044	
			2D	13	001C8		BEQL	35\$	
0000042C	8F		50	D1	001CA		CMPL	R0, #1068	
			24	13	001D1		BEQL	35\$	
00000434	8F		50	D1	001D3		CMPL	R0, #1076	
			1B	13	001DA		BEQL	35\$	
0000043C	8F		50	D1	001DC		CMPL	R0, #1084	
			12	13	001E3		BEQL	35\$	
0000044C	8F		50	D1	001E5		CMPL	R0, #1100	
			09	13	001EC		BEQL	35\$	
00000454	8F		50	D1	001EE		CMPL	R0, #1108	
			05	12	001F5		BNEQ	36\$	
	53		02	D0	001F7	35\$:	MOVL	#2, TRAP_OR_FAULT	: 6705
			03	11	001FA		BRB	37\$	
	53		01	D0	001FC	36\$:	MOVL	#1, TRAP_OR_FAULT	: 6721
	55		69	D0	001FF	37\$:	MOVL	DBG\$RUNFRAME+64, ADDRESS	: 6726
		00C5	C6	9F	00202		PUSHAB	P.ADG	: 6731
	67		01	FB	00206		CALLS	#1, DBG\$PRINT	
	01		53	D1	00209		CMPL	TRAP_OR_FAULT, #1	: 6732
			06	12	0020C		BNEQ	38\$	
		00D4	C6	9F	0020E		PUSHAB	P.ADH	: 6734
			04	11	00212		BRB	39\$	
		00DF	C6	9F	00214	38\$:	PUSHAB	P.ADI	: 6737
	67		01	FB	00218	39\$:	CALLS	#1, DBG\$PRINT	
		01	A4	95	0021B	40\$:	TSTB	1(R4)	: 6748
			0C	12	0021E		BNEQ	41\$	
	02	02	A4	91	00220		CMPB	2(R4), #2	: 6749
			06	12	00224		BNEQ	41\$	
	55	04	A8	D0	00226		MOVL	ACCVIO_PC, ADDRESS	: 6751
			03	11	0022A		BRB	42\$	
	55		69	D0	0022C	41\$:	MOVL	DBG\$RUNFRAME+64, ADDRESS	: 6754
			52	DD	0022F	42\$:	PUSHL	R2	: 6759
			55	DD	00231		PUSHL	ADDRESS	
0000V	CF		02	FB	00233		CALLS	#2, PRINT_PC_AND_INSTRUCTION	
	6A		00	FB	00238		CALLS	#0, DBG\$NEWLINE	: 6760
	02	02	A4	91	0023B		CMPB	2(R4), #2	: 6766

0A	18	A2	0089	03	13	0023F	BEQL	43\$:
10	18	A2	04	E1	00241	BRW	49\$:	:
12	18	A2	05	E0	00244	BBC	#4, 24(R2), 44\$:	6774
		50	04	E0	00249	BBS	#5, 24(R2), 45\$:	6775
07		50	00000000G	00	E0	0024E	BBS	#4, 24(R2), 46\$	6776
		50	00000000G	0A	E1	00253	MOVL	DBG\$GB_STP_PTR, R0	6777
		50	00000000G	0A	E1	0025A	BBC	#10, (R0), -46\$:
	0000V	CF		55	DD	0025E	PUSHL	ADDRESS	6779
			00E3	01	FB	00260	CALLS	#1, PRINT_SOURCE	:
		67		C6	9F	00265	PUSHAB	P.ADJ	6784
				01	FB	00269	CALLS	#1, DBG\$PRINT	:
				7E	D4	0026C	CLRL	-(SP)	6785
			00000000G	00	DD	0026E	PUSHL	DBG\$GL_SIGN_FLAG	6786
		7E	00000000G	00	9A	00274	MOVZBL	DBG\$GB_RADIX+2, -(SP)	6785
			FC	A8	DD	0027B	PUSHL	OLDVALUE	:
00000000G		00		04	FB	0027E	CALLS	#4, DBG\$PRINT VALUE	:
		6A		00	FB	00285	CALLS	#0, DBG\$NEWLINE	6787
			00F2	C6	9F	00288	PUSHAB	P.ADK	6788
		67		01	FB	0028C	CALLS	#1, DBG\$PRINT	:
				7E	D4	0028F	CLRL	-(SP)	6789
			00000000G	00	DD	00291	PUSHL	DBG\$GL_SIGN_FLAG	6790
		7E	00000000G	00	9A	00297	MOVZBL	DBG\$GB_RADIX+2, -(SP)	6789
			F8	A8	DD	0029E	PUSHL	NEWVALUE	:
00000000G		00		04	FB	002A1	CALLS	#4, DBG\$PRINT VALUE	:
		6A		00	FB	002A8	CALLS	#0, DBG\$NEWLINE	6791
		01		64	91	002AB	CMPB	(R4), #1	6796
				07	13	002AE	BEQL	47\$:
		50	0101	C6	9E	002B0	MOVAB	P.ADL, R0	6799
				05	11	002B5	BRB	48\$:
		50	010B	C6	9E	002B7	MOVAB	P.ADM, R0	6801
				50	DD	002BC	PUSHL	R0	:
		67		01	FB	002BE	CALLS	#1, DBG\$PRINT	6796
				52	DD	002C1	PUSHL	R2	6806
				69	DD	002C3	PUSHL	DBG\$RUNFRAME+64	:
	0000V	CF		02	FB	002C5	CALLS	#2, PRINT_PC_AND_INSTRUCTION	:
		6A		00	FB	002CA	CALLS	#0, DBG\$NEWLINE	6807
		23		68	E9	002CD	BLBC	TYPE_SOURCE, 52\$	6813
0A	18	A2		04	E1	002D0	BBC	#4, 24(R2), 50\$	6814
10	18	A2		05	E0	002D5	BBS	#5, 24(R2), 51\$	6815
14	18	A2		04	E0	002DA	BBS	#4, 24(R2), 52\$	6816
		50	00000000G	00	DD	002DF	MOVL	DBG\$GB_STP_PTR, R0	6817
09		60		0A	E1	002E6	BBC	#10, (R0), -52\$:
				69	DD	002EA	PUSHL	DBG\$RUNFRAME+64	6820
	0000V	CF		01	FB	002EC	CALLS	#1, PRINT_SOURCE	:
				68	D4	002F1	CLRL	TYPE_SOURCE	6821
				04	04	002F3	RET		6829

; Routine Size: 756 bytes, Routine Base: DBG\$CODE + 25EA


```
: 6717 6830 1 ROUTINE INSQUE1(ENTRY_ADDRESS, QUEUE_ADDRESS): NOVALUE =
: 6718 6831 1
: 6719 6832 1 FUNCTION
: 6720 6833 1 This routine adds an entry into a queue. This routine is used when
: 6721 6834 1 the forward and backward links are not the 0th and 1st longwords in
: 6722 6835 1 the queue head and queue entry, but the 2nd and 3rd.
: 6723 6836 1
: 6724 6837 1 INPUTS
: 6725 6838 1 QUEUE_ADDRESS : Pointer to queue head.
: 6726 6839 1 ENTRY_ADDRESS : Pointer to entry.
: 6727 6840 1
: 6728 6841 1 OUTPUTS
: 6729 6842 1 None
: 6730 6843 1
: 6731 6844 2 BEGIN
: 6732 6845 2
: 6733 6846 2 MAP
: 6734 6847 2 ENTRY_ADDRESS : REF QUEUE_HEAD,
: 6735 6848 2 QUEUE_ADDRESS : REF QUEUE_HEAD;
: 6736 6849 2
: 6737 6850 2 LOCAL
: 6738 6851 2 NEXTS_ADDRESS : REF QUEUE_HEAD; ! Next entry's pointer.
: 6739 6852 2
: 6740 6853 2
: 6741 6854 2 ! Point to the current 1st (next) entry.
: 6742 6855 2
: 6743 6856 2 NEXTS_ADDRESS = .QUEUE_ADDRESS [L_QUEUE_FLINK1];
: 6744 6857 2
: 6745 6858 2
: 6746 6859 2 ! Link the queue entry into the queue at the head.
: 6747 6860 2
: 6748 6861 2 ENTRY_ADDRESS [L_QUEUE_FLINK1] = .QUEUE_ADDRESS [L_QUEUE_FLINK1];
: 6749 6862 2 ENTRY_ADDRESS [L_QUEUE_BLINK1] = .QUEUE_ADDRESS;
: 6750 6863 2
: 6751 6864 2
: 6752 6865 2 ! Link the queue to the queue entry.
: 6753 6866 2
: 6754 6867 2 QUEUE_ADDRESS [L_QUEUE_FLINK1] = .ENTRY_ADDRESS;
: 6755 6868 2 NEXTS_ADDRESS [L_QUEUE_BLINK1] = .ENTRY_ADDRESS;
: 6756 6869 1 END;
```

			0004	00000	INSQUE1: .WORD	Save R2	
	50	08	AC	D0	00002	MOVL	QUEUE_ADDRESS, R0
	52	08	A0	D0	00006	MOVL	8(R0), NEXTS_ADDRESS
	51	04	AC	D0	0000A	MOVL	ENTRY_ADDRESS, R1
08	A1	08	A0	D0	0000E	MOVL	8(R0), 8(R1)
0C	A1		50	D0	00013	MOVL	R0, 12(R1)
08	A0		51	D0	00017	MOVL	R1, 8(R0)
0C	A2		51	D0	0001B	MOVL	R1, 12(NEXTS_ADDRESS)
			04	0001F	RET		

```
: 6830
: 6856
: 6861
: 6862
: 6867
: 6868
: 6869
```

; Routine Size: 32 bytes, Routine Base: DBG\$CODE + 28DE

DBGEVENT
V04-000

E 4
16-Sep-1984 00:59:10
14-Sep-1984 12:16:53

VAX-11 Bliss-32 V4.0-742
[DEBUG.SRC]DBGEVENT.B32;1

Page 190
(21)


```
: 6758      6870 1 ROUTINE REMQUE1(ENTRY_ADDRESS): NOVALUE =
: 6759      6871 1
: 6760      6872 1 FUNCTION
: 6761      6873 1     This routine deletes an entry from a queue. This routine is used when
: 6762      6874 1     the forward and backward links are not the 0th and 1st longwords in
: 6763      6875 1     the queue head and queue entry, but the 2nd and 3rd.
: 6764      6876 1
: 6765      6877 1 INPUTS
: 6766      6878 1     ENTRY_ADDRESS :      Pointer to entry.
: 6767      6879 1
: 6768      6880 1 OUTPUTS
: 6769      6881 1     None
: 6770      6882 1
: 6771      4883 2 BEGIN
: 6772      6883 2
: 6773      6885 2 MAP
: 6774      6886 2     ENTRY_ADDRESS : REF QUEUE_HEAD;
: 6775      6887 2
: 6776      6888 2 LOCAL
: 6777      6889 2     NEXTS_ADDRESS : REF QUEUE_HEAD,
: 6778      6890 2     QUEUE_ADDRESS : REF QUEUE_HEAD;
: 6779      6891 2
: 6780      6892 2
: 6781      6893 2     ! Point to the queue head and next entry.
: 6782      6894 2     !
: 6783      6895 2     QUEUE_ADDRESS = .ENTRY_ADDRESS [L_QUEUE_BLINK1];
: 6784      6896 2     NEXTS_ADDRESS = .ENTRY_ADDRESS [L_QUEUE_FLINK1];
: 6785      6897 2
: 6786      6898 2
: 6787      6899 2     ! Unlink the entry from the queue.
: 6788      6900 2     !
: 6789      6901 2     QUEUE_ADDRESS [L_QUEUE_FLINK1] = .NEXTS_ADDRESS;
: 6790      6902 2     NEXTS_ADDRESS [L_QUEUE_BLINK1] = .QUEUE_ADDRESS;
: 6791      6903 1 END;
```

			0000	00000	REMQUE1: .WORD	Save nothing	
	50	04	AC	D0	00002	ENTRY_ADDRESS, R0	
	50	08	A0	7D	00006	8(R0); NEXTS_ADDRESS	
08	A1		50	D0	0000A	NEXTS_ADDRESS, 8(QUEUE_ADDRESS)	
0C	A0		51	D0	0000E	QUEUE_ADDRESS, 12(NEXTS_ADDRESS)	
			04	00012	RET		

```
: 6870
: 6895
: 6896
: 6901
: 6902
: 6903
```

; Routine Size: 19 bytes, Routine Base: DBG\$CODE + 28FE

```

: 6793      6904 1 ROUTINE PRINT_SOURCE(ADDRESS): NOVALUE =
: 6794      6905 2 BEGIN
: 6795      6906 2 BUILTIN
: 6796      6907 2 FP;
: 6797      6908 2
: 6798      6909 2 .FP = DBG$FINAL_HANDL;
: 6799      6910 2 DBG$SRC_TYPE_PC_SOURCE (.ADDRESS, .ADDRESS, 0, TRUE);
: 6800      6911 1 END;

```

```

                                0000 00000 PRINT_SOURCE:
                                .WORD
6D 00000000G 00 9E 00002      MOVAB Save nothing
                                01 DD 00009      DBG$FINAL_HANDL, (FP)
                                7E D4 0000B      PUSHL #1
                                04 AC DD 0000D      LLRL -(SP)
                                04 AC DD 00010      PUSHL ADDRESS
                                04 FB 00013      PUSHL ADDRESS
                                04 0001A      CALLS #4, DBG$SRC_TYPE_PC_SOURCE
                                RET
00000000G 00

```

```

: 6904
: 6909
: 6910
:
:
: 6911

```

; Routine Size: 27 bytes, Routine Base: DBG\$CODE + 2911


```

: 6802      6912 1 ROUTINE PRINT_PC_AND_INSTRUCTION(ADDRESS, EVENT): NOVALUE =
: 6803      6913   BEGIN
: 6804      6914   MAP
: 6805      6915       ADDRESS :      REF VECTOR [, BYTE],
: 6806      6916       EVENT  :      REF EVENT$EVENT_DESCRIPTOR;
: 6807      6917
: 6808      6918   LOCAL ADDRESS_2;
: 6809      6919   BUILTIN
: 6810      6920       FP;
: 6811      6921
: 6812      6922
: 6813      6923   ! Establish a handler for errors. This is in case the instruction
: 6814      6924   ! we have stepped to is invalid, and the INS_DECODE routine signals
: 6815      6925   ! an error.
: 6816      6926
: 6817      6927   .FP = PRINT_PC_INST_HANDLER;
: 6818      6928
: 6819      6929
: 6820      6930   ! Display 'ROUTINE' if the current address points after an entry point.
: 6821      6931   !
: 6822      6932   IF .EVENT [EVENT$V_BREAK_ADDRESS]
: 6823      6933   THEN
: 6824      6934       DBG$PRINT ($AC ('routine '))
: 6825      6935
: 6826      6936   ELSE
: 6827      6937       BEGIN
: 6828      6938       IF DBG$IS_IT_ENTRY ((ADDRESS_2 = .ADDRESS - 2))
: 6829      6939       THEN
: 6830      6940           DBG$PRINT ($AC ('routine '))
: 6831      6941       ELSE
: 6832      6942           ADDRESS_2 = .ADDRESS;
: 6833      6943       END;
: 6834      6944
: 6835      6945
: 6836      6946   ! Print the address, but symbolize it first.
: 6837      6947   !
: 6838      6948   IF .EVENT [EVENT$V_BREAK_ADDRESS]
: 6839      6949   THEN
: 6840      6950       DBG$PRINT_IDENTIFIER (.EVENT [EVENT$L_PRIMARY])
: 6841      6951   ELSE
: 6842      6952       DBG$PRINT_IDENTIFIER_PC (.ADDRESS_2);
: 6843      6953
: 6844      6954
: 6845      6955   ! Display the instruction we're at, if we're not line stepping.
: 6846      6956   !
: 6847      6957   IF .DBG$GB_STP_PTR [EVENT$V_STEPPING_LINE] THEN RETURN;
: 6848      6958
: 6849      6959   DBG$PRINT($AC (': '));
: 6850      6960   DBG$INS_DECODE (.ADDRESS, TRUE, FALSE);
: 6851      6961   END;

```

.PSECT DBG\$PLIT,NOWRT, SHR, PIC,0

```

20 65 6E 69 74 75 6F 72 08 00392 P.ADN: .ASCII <8>\routine \
20 65 6E 69 74 75 6F 72 08 0039B P.ADO: .ASCII <8>\routine \

```

20 3A 02 003A4 P.ADP: .ASCII <2>\: \

.PSECT DBG\$CODE,NOWRT, SHR, PIC,0

				003C 00000	PRINT_PC AND INSTRUCTION:		
		55	00000000G	00	9E 00002	WORD Save R2,R3,R4,R5	: 6912
		54	00000000'	EF	9E 00009	MOVAB DBG\$PRINT, R5	: 6913
		6D	0000V	CF	9E 00010	MOVAB P.ADN, R4	: 6927
		52	08	AC	D0 00015	MOVAB PRINT_PC_INST_HANDLER, (FP)	: 6932
04	19	A2		05	E1 00019	MOVL EVENT, R2	: 6933
				54	DD 0001E	BBC #5, 25(R2), 1\$: 6934
				14	11 00020	PUSHL R4	: 6938
53	04	AC		02	C3 00022 1\$:	BRB 2\$: 6938
				53	DD 00027	SUBL3 #2, ADDRESS, ADDRESS_2	
		00000000G	00	01	FB 00029	PUSHL ADDRESS_2	
			08	50	E9 00030	CALLS #1, DBG\$IS_IT_ENTRY	
				09	A4 9F 00033	BLBC R0, 3\$	
		65		01	FB 00036 2\$:	PUSHAB P.ADN	: 6940
				04	11 00039	CALLS #1, DBG\$PRINT	
				04	AC D0 0003B 3\$:	BRB 4\$	
0C	19	A2		05	E1 0003F 4\$:	MOVL ADDRESS, ADDRESS_2	: 6942
				28	A2 DD 00044	BBC #5, 25(R2), 5\$: 6948
		00000000G	00	01	FB 00047	PUSHL 40(R2)	: 6950
				09	11 0004E	CALLS #1, DBG\$PRINT_IDENTIFIER	
				53	DD 00050 5\$:	BRB 6\$	
		00000000G	00	01	FB 00052	PUSHL ADDRESS_2	: 6952
				50	00000000G 00 D0 00059 6\$:	CALLS #1, DBG\$PRINT_IDENTIFIER_PC	
			13	01	A0 E8 00060	MOVL DBG\$GB_STP_PTR, R0	: 6957
				12	A4 9F 00064	BLBS 1(R0), -7\$	
		65		01	FB 00067	PUSHAB P.ADP	: 6959
		7E		01	7D 0006A	CALLS #1, DBG\$PRINT	
				04	AC DD 0006D	MOVQ #1, -(SP)	: 6960
		00000000G	00	03	FB 00070	PUSHL ADDRESS	
				04	00077 7\$:	CALLS #3, DBG\$INS_DECODE	: 6961
						RET	

; Routine Size: 120 bytes, Routine Base: DBG\$CODE + 292C


```
: 6853      6962 1 ROUTINE PRINT_PC_INST_HANDLER(SIG_ARG_LIST) =
: 6854      6963 1
: 6855      6964 1 FUNCTION
: 6856      6965 1     Handler for errors that occur during the PRINT_PC AND INSTRUCTION
: 6857      6966 1     routine. We need a handler so that after displaying the error,
: 6858      6967 1     we will unwind to the caller of PRINT_PC AND INSTRUCTION, and
: 6859      6968 1     continue with any eventpoint processing that was being done.
: 6860      6969 1
: 6861      6970 1 INPUTS
: 6862      6971 1     SIG_ARG_LIST - The signal argument list that we get from
: 6863      6972 1     the condition handling facility.
: 6864      6973 1
: 6865      6974 1 OUTPUTS
: 6866      6975 1
: 6867      6976 2 BEGIN
: 6868      6977 2 MAP
: 6869      6978 2     SIG_ARG_LIST: REF BLOCK[,BYTE];
: 6870      6979 2
: 6871      6980 2
: 6872      6981 2     ! If informational just resignal
: 6873      6982 2
: 6874      6983 2 IF .SIG_ARG_LIST[CHFSL_SIG_NAME]
: 6875      6984 2 THEN
: 6876      6985 2     RETURN SSS_RESIGNAL;
: 6877      6986 2
: 6878      6987 2
: 6879      6988 2     ! If we are in the middle of an unwind then continue unwinding.
: 6880      6989 2
: 6881      6990 2 IF .SIG_ARG_LIST[CHFSL_SIG_NAME] EQL SSS_UNWIND
: 6882      6991 2 THEN
: 6883      6992 2     RETURN SSS_CONTINUE;
: 6884      6993 2
: 6885      6994 2
: 6886      6995 2     ! Otherwise, display a message saying that an error occurred during
: 6887      6996 2     instruction decode and then display the actual error that occurred.
: 6888      6997 2
: 6889      6998 2 SIGNAL(DBG$ERRINSDEC);
: 6890      6999 2 DBG$PUTMSG(.SIG_ARG_LIST);
: 6891      7000 2
: 6892      7001 2
: 6893      7002 2     ! Then unwind to the routine that called PRINT_PC_AND_INSTRUCTION.
: 6894      7003 2
: 6895      7004 2 SETUNWIND();
: 6896      7005 2 RETURN SSS_CONTINUE;
: 6897      7006 1 END;
```

```
0004 0000 PRINT_PC_INST_HANDLER:
                                .WORD Save R2
                                MOVL SIG_ARG_LIST, R2
                                BLBC 4(R2), T$
                                MOVZWL #2328, R0
                                RET
00000920 8F 04 A2 D1 00010 1$: CMPL 4(R2), #2336
```

```
: 6962
: 6983
: 6985
: 6990
```

DBGEVENT
V04-000

K 4
16-Sep-1984 00:59:10
14-Sep-1984 12:16:53

VAX-11 Bliss-32 V4.0-742
[DEBUG.SRC]DBGEVENT.B32;1

Page 196
(25)

00000000G	00	000280AB	1F	13	00018	BEQL	2\$:	6998
			8F	DD	0001A	PUSHL	#164011	:	
			01	FB	00020	CALLS	#1, LIB\$SIGNAL	:	
00000000G	00		52	DD	00027	PUSHL	R2	:	6999
			01	FB	00029	CALLS	#1, DBG\$PUTMSG	:	
00000000G	00		7E	7C	00030	CLRQ	-(SP)	:	7004
			02	FB	00032	CALLS	#2, SYSSUNWIND	:	
	50		01	DD	00039	MOVL	#1, R0	:	7005
			04	0003C	2\$:	RET		:	7006

; Routine Size: 61 bytes, Routine Base: DBG\$CODE + 29A4


```

6899 7007 1 ROUTINE SHOW_EVENT_ENTRY(EVENT_ENTRY): NOVALUE =
6900 7008 1
6901 7009 1 FUNCTION
6902 7010 1     This routine displays an event entry as in:
6903 7011 1
6904 7012 1     SHOW [ BREAK : TRACE : WATCH ]
6905 7013 1
6906 7014 1 INPUTS
6907 7015 1     EVENT_ENTRY :           Pointer to event entry.
6908 7016 1
6909 7017 1 OUTPUTS
6910 7018 1     Displays the event entry via DBG$PRINT.
6911 7019 1
6912 7020 1
6913 7021 2 BEGIN
6914 7022 2
6915 7023 2 MAP
6916 7024 2     EVENT_ENTRY: REF EVENT$EVENT_DESCRIPTOR; ! Event Entry Pointer
6917 7025 2
6918 7026 2
6919 7027 2 LOCAL
6920 7028 2     VMS_DESC: DBG$STG_DESC,
6921 7029 2     VAL_DESC: REF DBG$VAL_DESC,
6922 7030 2     WHEN_ENTRY:
6923 7031 2         REF EVENT$WHEN_DESCRIPTOR, ! When Entry Pointer
6924 7032 2     DO_LIST_ENTRY:
6925 7033 2         REF EVENT$DO_LIST_DESCRIPTOR, ! DO List Entry Pointer
6926 7034 2     STRING_DESCRIPTOR: BLOCK[8,BYTE], ! String descriptor
6927 7035 2     OUTPUT_BUFFER:
6928 7036 2         VECTOR[TTY_OUT_WIDTH,BYTE], ! Temporary formatting buffer
6929 7037 2     DUMMY;
6930 7038 2         ! Dummy routine argument
6931 7039 2
6932 7040 2
6933 7041 2 ! Skip STEPS, SKIPS, and IGNOR entries.
6934 7042 2
6935 7043 2 IF .EVENT_ENTRY [EVENT$B_CMD_TYPE] EQLU EVENT$K_TYPE_STEPS OR
6936 7044 2     .EVENT_ENTRY [EVENT$B_CMD_TYPE] EQLU EVENT$K_TYPE_SKIPS OR
6937 7045 2     .EVENT_ENTRY [EVENT$B_CMD_TYPE] EQLU EVENT$K_TYPE_IGNORE OR
6938 7046 2     .EVENT_ENTRY [EVENT$B_CMD_TYPE] EQLU EVENT$K_TYPE_SSINT
6939 7047 2 THEN
6940 7048 2     RETURN;
6941 7049 2
6942 7050 2
6943 7051 2 ! Setup a string descriptor.
6944 7052 2
6945 7053 2 STRING_DESCRIPTOR [DSC$B_DTYPE] = DSC$K_DTYPE_T;
6946 7054 2 STRING_DESCRIPTOR [DSC$B_CLASS] = DSC$K_CLASS_S;
6947 7055 2 STRING_DESCRIPTOR [DSC$A_POINTER] = 0;
6948 7056 2 STRING_DESCRIPTOR [DSC$W_LENGTH] = 0;
6949 7057 2
6950 7058 2
6951 7059 2 ! For watchpoint events, see whether the watched Primary
6952 7060 2 ! is still valid. If not, convert the Primary to a Volatile
6953 7061 2 ! Value Descriptor and display an informational to that effect.
6954 7062 2
6955 7063 2 IF .EVENT_ENTRY[EVENT$B_SUB_KIND] EQL EVENT$K_ACC_MDFY
```

```

: 6956      7064      2
: 6957      7065
: 6958      7066
: 6959      7067
: 6960      7068
: 6961      7069
: 6962      7070
: 6963      7071
: 6964      7072
: 6965      7073
: 6966      7074
: 6967      7075
: 6968      7076
: 6969      7077
: 6970      7078
: 6971      7079
: 6972      7080
: 6973      7081
: 6974      7082
: 6975      7083
: 6976      7084
: 6977      7085
: 6978      7086
: 6979      7087
: 6980      7088
: 6981      7089
: 6982      7090
: 6983      7091
: 6984      7092
: 6985      7093
: 6986      7094  4
: 6987      7095  4
: 6988      7096  4
: 6989      7097  4
: 6990      7098  4
: 6991      7099  4
: 6992      7100  4
: 6993      7101  4
: 6994      7102  4
: 6995      7103  4
: 6996      7104  4
: 6997      7105  4
: 6998      7106  4
: 6999      7107  4
: 7000      7108  4
: 7001      7109  4
: 7002      7110  4
: 7003      7111  4
: 7004      7112  4
: 7005      7113  4
: 7006      7114  4
: 7007      7115  4
: 7008      7116  4
: 7009      7117  4
: 7010      7118  4
: 7011      7119  4
: 7012      7120  4

THEN
  BEGIN
    LOCAL
      PRIMARY,
      SYMID_LIST,
      VALDESC,
      V_VALDESC: REF DBG$VALDESC;

    ! Copy the Primary since PRIM_TO_VAL sometimes has the side effect
    ! of modifying its argument. This copy is done into temporary
    ! memory (fourth parameter is FALSE). Then call PRIM_TO_VAL
    ! and get a Volatile Value Descriptor (this will have
    ! the real address of the watched variable embedded in it).
    ! PRIM_TO_VAL is a cover routine for DBG$PRIM_TO_VAL; it has
    ! a handler which sets INVALID_FLAG if the Primary is no longer
    ! valid (has gone out of scope).
    DBG$NCOPY_DESC (.EVENT_ENTRY[EVENT$$_PRIMARY], PRIMARY, DUMMY, FALSE);
    INVALID_FLAG = FALSE;
    PRIM_TO_VAL (.PRIMARY, DBG$K_V_VALUE_DESC, V_VALDESC);

    ! Check whether the Primary is still valid, i.e., whether it still
    ! points to the same storage as it did when the watchpoint was set.
    IF .INVALID_FLAG OR
        CH$NEQ (T2, V_VALDESC[DBG$A_VALUE_VMSDESC],
                12, EVENT_ENTRY[EVENT$$_VMSDESC])
    THEN
      BEGIN

        ! Print an informational saying the Primary is invalid.
        DBG$PRINT ($AC ('%DEBUG-I-WATCHVAR, watched variable '));
        DBG$PRINT IDENTIFIER(.EVENT_ENTRY[EVENT$$_PRIMARY]);
        IF .INVALID_FLAG
        THEN
          DBG$PRINT ($AC (' has gone out of scope'))
        ELSE
          DBG$PRINT ($AC (' now points to a different address'));
        DBG$NEWLINE();

        ! Free up the Primary.
        IF DBG$NGET_SYMID (.EVENT_ENTRY[EVENT$$_PRIMARY], SYMID_LIST, DUMMY)
        THEN
          DBG$STA_UNLOCK_SYMID (.SYMID_LIST);
          DBG$NFREE_DESC (.EVENT_ENTRY[EVENT$$_PRIMARY], DUMMY);

        ! Replace it with the Volatile Value Descriptor.
        V_VALDESC = DBG$MAKE_VAL_DESC(EVENT_ENTRY[EVENT$$_VMSDESC], DBG$K_V_VALUE_DESC);
        IF DBG$NGET_SYMID (.V_VALDESC, SYMID_LIST, DUMMY)
```



```

: 7013      7121  4      THEN
: 7014      7122  4      DBGSSTA LOCK_SYMID (.SYMID_LIST);
: 7015      7123  4      DBG$NCOPI_DESC (.V_VALDESC, EVENT_ENTRY[EVENT$B_PRIMARY], DUMMY, TRUE);
: 7016      7124  3      END;
: 7017      7125  2      END;
: 7018      7126  2
: 7019      7127  2
: 7020      7128  2      ! Print the event type: breakpoint, tracepoint, or watchpoint.
: 7021      7129  2      !
: 7022      7130  2      SELECTONE .EVENT_ENTRY[EVENT$B_CMD_TYPE] OF
: 7023      7131  2      SET
: 7024      7132  2
: 7025      7133  2      [EVENT$K_TYPE_BREAK]:
: 7026      7134  2      DBGS$PRINT($AC('breakpoint'));
: 7027      7135  2
: 7028      7136  2      [EVENT$K_TYPE_TRACE]:
: 7029      7137  2      DBGS$PRINT($AC('tracepoint'));
: 7030      7138  2
: 7031      7139  2      [EVENT$K_TYPE_WATCH]:
: 7032      7140  2      DBGS$PRINT($AC('watchpoint'));
: 7033      7141  2
: 7034      7142  2      TES;
: 7035      7143  2
: 7036      7144  2
: 7037      7145  2      ! Display the event kind (access, exception, instruction).
: 7038      7146  2      !
: 7039      7147  2      SELECTONE .EVENT_ENTRY [EVENT$B_CMD_KIND] OF
: 7040      7148  2      SET
: 7041      7149  2
: 7042      7150  2
: 7043      7151  2      ! This is an access (read, write, modify, execute) kind
: 7044      7152  2      ! of event point. Display the address.
: 7045      7153  2      !
: 7046      7154  2      [EVENT$K_KIND_ACC]:
: 7047      7155  3      BEGIN
: 7048      7156  3
: 7049      7157  3
: 7050      7158  3      ! Print the access kind if read, write, or modify. If the address
: 7051      7159  3      ! is an entry point, indicate that it is a routine.
: 7052      7160  3      !
: 7053      7161  3      IF .EVENT_ENTRY[EVENT$B_CMD_TYPE] NEQU EVENT$K_TYPE_WATCH
: 7054      7162  3      THEN
: 7055      7163  4      BEGIN
: 7056      7164  4      SELECTONE .EVENT_ENTRY[EVENT$B_SUB_KIND] OF
: 7057      7165  4      SET
: 7058      7166  4
: 7059      7167  4      [EVENT$K_ACC_READ]:
: 7060      7168  4      DBGS$PRINT($AC(' on read of '));
: 7061      7169  4
: 7062      7170  4      [EVENT$K_ACC_WRIT]:
: 7063      7171  4      DBGS$PRINT($AC(' on write of '));
: 7064      7172  4
: 7065      7173  4      [EVENT$K_ACC_MDFY]:
: 7066      7174  4      DBGS$PRINT($AC(' on modify of '));
: 7067      7175  4
: 7068      7176  4      [EVENT$K_ACC_EXEC]:
: 7069      7177  4      DBGS$PRINT($AC(' at '));
```

```
: 7070 7178 4
: 7071 7179 4
: 7072 7180 4
: 7073 7181 4
: 7074 7182 4
: 7075 7183 4
: 7076 7184 4
: 7077 7185 4
: 7078 7186 4
: 7079 7187 4
: 7080 7188 4
: 7081 7189 3
: 7082 7190 3
: 7083 7191 3
: 7084 7192 3
: 7085 7193 3
: 7086 7194 3
: 7087 7195 3
: 7088 7196 3
: 7089 7197 3
: 7090 7198 3
: 7091 7199 3
: 7092 7200 3
: 7093 7201 3
: 7094 7202 3
: 7095 7203 3
: 7096 7204 3
: 7097 7205 3
: 7098 7206 3
: 7099 7207 3
: 7100 7208 3
: 7101 7209 3
: 7102 7210 4
: 7103 7211 4
: 7104 7212 4
: 7105 7213 4
: 7106 7214 4
: 7107 7215 4
: 7108 7216 4
: 7109 7217 4
: 7110 7218 4
: 7111 7219 3
: 7112 7220 3
: 7113 7221 2
: 7114 7222 2
: 7115 7223 2
: 7116 7224 2
: 7117 7225 2
: 7118 7226 2
: 7119 7227 3
: 7120 7228 3
: 7121 7229 3
: 7122 7230 3
: 7123 7231 3
: 7124 7232 3
: 7125 7233 3
: 7126 7234 4
```

```
[EVENT$K_ACC_RTRN]:
  DBG$PRINT($AC(' on return from '));

TES;

END

! For watchpoints, simply say 'watchpoint of'.
ELSE
  DBG$PRINT($AC(' of '));

! We print the address next. If the address is a routine address
! (a CALLS/CALLG entry point), we refer to it as "routine XXX",
! where XXX is the routine name. If it is not a routine, we omit
! the "routine" designation.
IF .EVENT_ENTRY[EVENT$V_BREAK_ADDRESS] OR
  DBG$IS_IT_ENTRY(.EVENT_ENTRY[EVENT$L_ADDRESS] - 2)
THEN
  DBG$PRINT($AC('routine '));

DBG$PRINT_IDENTIFIER(.EVENT_ENTRY[EVENT$L_PRIMARY]);

! Check for the /SOURCE or /NOSOURCE override switch. If set, we
! print "[source]" or "[nosource]", as appropriate.
IF .EVENT_ENTRY[EVENT$V_OVRD_SOURCE]
THEN
  BEGIN
    IF .EVENT_ENTRY[EVENT$V_STEP_SOURCE]
    THEN
      DBG$PRINT($AC(' [source]'));
    ELSE
      DBG$PRINT($AC(' [nosource]'));
  END;
END;

! End of case for access kind of event

! This is an instruction type.
[EVENT$K_KIND_INS]:
  BEGIN

! Display the /SOURCE or /NOSOURCE qualifier if specified.
IF .EVENT_ENTRY[EVENT$V_OVRD_SOURCE]
THEN
  BEGIN
```



```
: 7127      7235  4      IF .EVENT_ENTRY[EVENT$V_STEP_SOURCE]
: 7128      7236  4      THEN
: 7129      7237  4          DBG$PRINT($AC('/source'))
: 7130      7238  4
: 7131      7239  4      ELSE
: 7132      7240  4          DBG$PRINT($AC('/nosource'));
: 7133      7241  4
: 7134      7242  4      END;
: 7135      7243  3
: 7136      7244  3
: 7137      7245  3      ! Display the /SOURCE or /NOSOURCE qualifier if specified.
: 7138      7246  3      !
: 7139      7247  3      IF .EVENT_ENTRY[EVENT$V_OVRD_NOSYSTEM]
: 7140      7248  3      THEN
: 7141      7249  4          BEGIN
: 7142      7250  4              IF .EVENT_ENTRY[EVENT$V_STEP_NOSYSTEM]
: 7143      7251  4              THEN
: 7144      7252  4                  DBG$PRINT($AC('/nosystem'))
: 7145      7253  4
: 7146      7254  4              ELSE
: 7147      7255  4                  DBG$PRINT($AC('/system'));
: 7148      7256  4
: 7149      7257  4          END;
: 7150      7258  3
: 7151      7259  3
: 7152      7260  3      ! Display the /OVER or /INTO qualifier if specified.
: 7153      7261  3      !
: 7154      7262  3      IF .EVENT_ENTRY[EVENT$V_OVRD_OVER]
: 7155      7263  3      THEN
: 7156      7264  4          BEGIN
: 7157      7265  4              IF .EVENT_ENTRY[EVENT$V_STEP_OVER]
: 7158      7266  4              THEN
: 7159      7267  4                  DBG$PRINT($AC('/over'))
: 7160      7268  4
: 7161      7269  4              ELSE
: 7162      7270  4                  DBG$PRINT($AC('/into'));
: 7163      7271  4
: 7164      7272  4          END;
: 7165      7273  3
: 7166      7274  3
: 7167      7275  3
: 7168      7276  3      ! Display the instruction type (calls, branches, all, or user
: 7169      7277  3      ! list.
: 7170      7278  3      !
: 7171      7279  3      DBG$PRINT      ($AC      (' on !AC'),
: 7172      7280  3      SELECTONE .EVENT_ENTRY [EVENT$B_SUB_KIND] OF
: 7173      7281  3          SET
: 7174      7282  3              [EVENT$K_INS_CALL]:
: 7175      7283  3                  SAC T'calls:');
: 7176      7284  3              [EVENT$K_INS_BRAN]:
: 7177      7285  3                  SAC T'branches:');
: 7178      7286  3              [EVENT$K_INS_EVR]:
: 7179      7287  3                  SAC T'instructions:');
: 7180      7288  3              [EVENT$K_INS_USER]:
: 7181      7289  3                  SAC T'instruction(s):');
: 7182      7290  3              [EVENT$K_INS_LINE]:
: 7183      7291  3                  SAC T'lines:');
:                                     TES
```

```
: 7184      7292      3
: 7185      7293      3
: 7186      7294      3
: 7187      7295      3
: 7188      7296      3
: 7189      7297      3
: 7190      7298      3
: 7191      7299      3
: 7192      7300      3
: 7193      7301      3
: 7194      7302      3
: 7195      7303      3
: 7196      7304      3
: 7197      7305      3
: 7198      7306      3
: 7199      7307      3
: 7200      7308      3
: 7201      7309      3
: 7202      7310      3
: 7203      7311      3
: 7204      7312      3
: 7205      7313      3
: 7206      7314      3
: 7207      7315      3
: 7208      7316      3
: 7209      7317      3
: 7210      7318      3
: 7211      7319      3
: 7212      7320      3
: 7213      7321      3
: 7214      7322      3
: 7215      7323      3
: 7216      7324      3
: 7217      7325      3
: 7218      7326      3
: 7219      7327      3
: 7220      7328      3
: 7221      7329      3
: 7222      7330      3
: 7223      7331      3
: 7224      7332      3
: 7225      7333      3
: 7226      7334      3
: 7227      7335      3
: 7228      7336      3
: 7229      7337      3
: 7230      7338      4
: 7231      7339      4
: 7232      7340      3
: 7233      7341      3
: 7234      7342      3
: 7235      7343      3
: 7236      7344      3
: 7237      7345      3
: 7238      7346      3
: 7239      7347      4
: 7240      7348      4

);

END;

! This is an exception type.
[EVENT$K KIND EXC]:
  DBG$PRINT($AC(' on exception'));

TES;

! If this entry is silent, say so.
IF .EVENT_ENTRY[EVENT$V_SILENT] THEN DBG$PRINT($AC(' [silent]'));

! If this entry is temporary, say so.
IF .EVENT_ENTRY[EVENT$V_ONCE_ONLY] THEN DBG$PRINT($AC(' [temporary]'));

! If this entry is an instruction type (/CALL, /BRANCH, etc.) and
! has an opcode bit map (i.e., is not for all instructions),
! display the opcode mnemonics indicated.
IF .EVENT_ENTRY [EVENT$B_CMD_KIND] EQLU EVENT$K_KIND_INS AND
.EVENT_ENTRY [EVENT$L_OPCODE_LIST] NEQA 0
THEN
  BEGIN

    ! Declare a pointer to an opcode list, and an opcode
    ! variable.
    LOCAL
      OPCODE_LIST: REF BITVECTOR [512],
      COLUMN;

    ! Point to the opcode list, and set the column to 0.
    OPCODE_LIST = .EVENT_ENTRY [EVENT$L_OPCODE_LIST];
    COLUMN = -1;
    INCR OPCODE FROM 0 TO 511 DO
      BEGIN
        IF .OPCODE_LIST [.OPCODE] THEN
          BEGIN
            LOCAL INDEX;
            COLUMN = (.COLUMN + 1) AND 7;
            IF .COLUMN EQLU 0 THEN DBG$NEWLINE();
            INDEX = .DBG$OPCODE_KIND_TABLE[.OPCODE];
            DBG$PRINT($AC('! !AD!'),-6,
                      DBG$OPCODE_NAME_TABLE[.INDEX,0,0,0,0]);
          END;
        END;
      END;
    END;
  END;
```



```
: 7241      7349      3
: 7242      7350      3
: 7243      7351      3
: 7244      7352      3
: 7245      7353      3
: 7246      7354      3
: 7247      7355      3
: 7248      7356      3
: 7249      7357      2
: 7250      7358      2
: 7251      7359      2
: 7252      7360      2
: 7253      7361      2
: 7254      7362      2
: 7255      7363      2
: 7256      7364      2
: 7257      7365      3
: 7258      7366      3
: 7259      7367      3
: 7260      7368      2
: 7261      7369      2
: 7262      7370      2
: 7263      7371      2
: 7264      7372      2
: 7265      7373      2
: 7266      7374      2
: 7267      7375      3
: 7268      7376      3
: 7269      7377      3
: 7270      7378      3
: 7271      7379      3
: 7272      7380      3
: 7273      7381      3
: 7274      7382      3
: 7275      7383      2
: 7276      7384      2
: 7277      7385      2
: 7278      7386      2
: 7279      7387      2
: 7280      7388      2
: 7281      7389      2
: 7282      7390      3
: 7283      7391      3
: 7284      7392      3
: 7285      7393      3
: 7286      7394      3
: 7287      7395      3
: 7288      7396      3
: 7289      7397      3
: 7290      7398      2
: 7291      7399      2
: 7292      7400      1
```

```
END;

DBG$NEWLINE();
END

! Otherwise, just print the line.
ELSE
  DBG$NEWLINE ();

! If the after count is not the default (1), print it.
IF .EVENT_ENTRY [EVENT$L_AFTER_COUNT] NEQU 1
THEN
  BEGIN
    DBG$PRINT($AC(' /after: !UL'), .EVENT_ENTRY[EVENT$L_AFTER_COUNT]);
    DBG$NEWLINE();
  END;

! If there is a WHEN entry, print it.
IF .EVENT_ENTRY [EVENT$L_WHEN] NEQA 0
THEN
  BEGIN
    LOCAL
      STRING : REF VECTOR [, WORD];

    WHEN ENTRY = .EVENT_ENTRY [EVENT$L_WHEN];
    STRING = .WHEN_ENTRY [EVENT$L_WHEN_POINT];
    DBG$PRINT($AC(' when (!AD)'), .STRING[0] - 1, STRING[1]);
    DBG$NEWLINE();
  END;

! If there is a DO list entry, print it.
IF .EVENT_ENTRY [EVENT$L_DO] NEQA 0
THEN
  BEGIN
    LOCAL
      STRING : REF VECTOR [, WORD];

    DO LIST_ENTRY = .EVENT_ENTRY [EVENT$L_DO];
    STRING = .DO_LIST_ENTRY [EVENT$L_DO_LIST_POINT];
    DBG$PRINT($AC(' do (!AD)'), .STRING[0] - 1, STRING[1]);
    DBG$NEWLINE();
  END;
END;
```

.PSECT DBG\$PLIT,NOWRT, SHR, PIC,0

```
48 43 54 41 57 2D 49 2D 47 55 42 45 44 25 24 003A7 P.ADQ: .ASCII \%DEBUG-I-WATCHVAR, watched variable \
61 76 20 64 65 68 63 74 61 77 20 2C 52 41 56 003B6
20 74 75 6F 20 65 6E 6F 67 20 73 61 68 20 16 003C5
6F 74 20 73 74 6E 69 6F 70 20 77 6F 6E 20 22 003CC P.ADR: .ASCII <22>\ has gone out of scope\
64 61 20 74 6E 65 72 65 66 66 69 64 20 61 20 003DB P.ADS: .ASCII \' now points to a different address\
74 6E 69 6F 70 6B 61 65 72 62 0A 00401
74 6E 69 6F 70 65 63 61 72 74 0A 00406 P.ADT: .ASCII <10>\breakpoint\
74 6E 69 6F 70 65 63 61 72 74 0A 00411 P.ADU: .ASCII <10>\tracepoint\
20 20 20 66 6F 20 65 74 61 65 72 20 0C 0041C P.ADV: .ASCII <10>\watchpoint\
66 6F 20 79 66 69 64 6F 6D 20 6E 6F 20 0D 00427 P.ADW: .ASCII <12>\ on read of \
20 66 6F 20 79 66 69 64 6F 6D 20 6E 6F 20 0E 00434 P.ADX: .ASCII <13>\ on write of \
6F 72 66 20 6E 72 75 74 65 72 20 6E 6F 20 0E 00442 P.ADY: .ASCII <14>\ on modify of \
00451 P.ADZ: .ASCII <4>\ at \
00456 P.AEA: .ASCII <16>\ on return from \
00465
00467 P.AEB: .ASCII <4>\ of \
0046C P.AEC: .ASCII <8>\routine \
00475 P.AED: .ASCII <9>\ [source]\
0047F P.AEE: .ASCII <11>\ [nosource]\
0048B P.AEF: .ASCII <7>\ /source\
00493 P.AEG: .ASCII <9>\ /nosource\
0049D P.AEH: .ASCII <9>\ /nosystem\
004A7 P.AEI: .ASCII <7>\ /system\
004AF P.AEJ: .ASCII <5>\ /over\
004B5 P.AEK: .ASCII <5>\ /into\
004BB P.AEL: .ASCII <7>\ on !AC\
004C3 P.AEM: .ASCII <6>\calls:\
004CA P.AEN: .ASCII <9>\branches:\
004D4 P.AEO: .ASCII <12>\instructions\
004E1 P.AEP: .ASCII <15>\instruction(s):\
004F0
004F1 P.AEQ: .ASCII <5>\lines\
004F7 P.AER: .ASCII <13>\ on exception\
00505 P.AES: .ASCII <9>\ [silent]\
0050F P.AET: .ASCII <12>\ [temporary]\
0051C P.AEU: .ASCII <5>\ !AD\
00522 P.AEV: .ASCII <14>\ /after: !UL\
00531 P.AEW: .ASCII <13>\ when (!AD)\
0053F P.AEX: .ASCII <11>\ do (!AD)\
```

.PSECT DBG\$CODE,NOWRT, SHR, PIC,0

OFFC 00000 SHOW_EVENT ENTRY:

```
SB 00000000G 00 9E 00002 .WORD Save R2,R3,R4,R5,R6,R7,R8,R9,R10,R11
SA 00000000G 00 9E 00009 MOVAB DBGS$GET_SYMID, R11
S9 00000000' EF 9E 0001C MOVAB DBGS$PRINT_IDENTIFIER, R10
S8 00000000G 00 9E 00017 MOVAB INVALID_FLAG, R9
S7 00000000G 00 9E 0001E MOVAB DBGS$NEWLINE, R8
S6 00000000' EF 9E 00025 MOVAB DBGS$PRINT, R7
SE FF58 CE 9E 0002C MOVAB P.ADQ, R6
S4 04 AC D0 00031 MOVAB -168(SP), SP
S5 14 A4 9E 00035 MOVL EVENT_ENTRY, R4
MOVAB 20(R4), R5
```

7007

7043

03	65	91	00039	CMPB	(R5), #3		
	0D	13	0003C	BEQL	1\$		
04	65	91	0003E	CMPB	(R5), #4	7044	
	08	13	00041	BEQL	1\$		
05	65	91	00043	CMPB	(R5), #5	7045	
	03	13	00046	BEQL	1\$		
06	65	91	00048	CMPB	(R5), #6	7046	
	01	12	0004B	BNEQ	2\$		
		04	0004D	RET			
EC	AD	010E0000	8F	D0	0004E	2\$:	
	FO		AD	D4	00056	MOVLE #17694720, STRING_DESCRIPTOR	
	02	02	A5	91	00059	CLRL STRING_DESCRIPTOR+4	
			03	13	0005D	CMPB 2(R5), #2	
		00B1	31	0005F	BEQL 3\$	7063	
		7E	D4	00062	BRW 9\$		
	10		AE	9F	00064	3\$: CLRL -(SP)	
	08		AE	9F	00067	PUSHAB DUMMY	
	28		A4	DD	0006A	PUSHAB PRIMARY	
00000000G	00		04	FB	0006D	PUSHL 40(R4)	
			69	D4	00074	CALLS #4, DBG\$NCPY_DESC	
			04	AE	9F	00076	CLRL INVALID_FLAG
	7E		83	8F	9A	00079	PUSHAB V_VALDESC
			08	AE	DD	0007D	MOVZBL #T31, -(SP)
0000V	CF		03	FB	00080	PUSHL PRIMARY	
	0C		69	E8	00085	CALLS #3, PRIM_TO_VAL	
	50		04	AE	D0	00088	BLBS INVALID_FLAG, 4\$
34	A4	14	0C	29	0008C	MOVL V_VALDESC, R0	
			7F	13	00092	CMPCC #T2, 20(R0), 52(R4)	
			56	DD	00094	BEQL 9\$	
			01	FB	00096	4\$: PUSHL R6	
67			A4	DD	00099	CALLS #1, DBG\$PRINT	
			01	FB	0009C	PUSHL 40(R4)	
6A			69	E9	0009F	CALLS #1, DBG\$PRINT_IDENTIFIER	
05			A6	9F	000A2	BLBC INVALID_FLAG, 5\$	
			03	11	000A5	PUSHAB P.ADR	
			A6	9F	000A7	BRB 6\$	
67			01	FB	000AA	5\$: PUSHAB P.ADS	
68			00	FB	000AD	6\$: CALLS #1, DBG\$PRINT	
			0C	AE	9F	000B0	CALLS #0, DBG\$NEWLINE
			0C	AE	9F	000B3	PUSHAB DUMMY
			28	A4	DD	000B6	PUSHAB SYMID_LIST
			03	FB	000B9	PUSHL 40(R4)	
6B			50	E9	000BC	CALLS #3, DBG\$NGET_SYMID	
0A			08	AE	DD	000BF	BLBC R0, 7\$
			01	FB	000C2	PUSHL SYMID_LIST	
00000000G	00		0C	AE	9F	000C9	CALLS #1, DBG\$STA_UNLOCK_SYMID
			28	A4	DD	000CC	PUSHAB DUMMY
00000000G	00		02	FB	000CF	PUSHL 40(R4)	
	7E		83	8F	9A	000D6	CALLS #2, DBG\$NFREE_DESC
			34	A4	9F	000DA	MOVZBL #131, -(SP)
00000000G	00		02	FB	000DD	PUSHAB 52(R4)	
	04	AE	50	D0	000E4	CALLS #2, DBG\$MAKE_VAL_DESC	
			0C	AE	9F	000E8	MOVL R0, V_VALDESC
			0C	AE	9F	000EB	PUSHAB DUMMY
			0C	AE	DD	000EE	PUSHAB SYMID_LIST
			03	FB	000F1	PUSHL V_VALDESC	
6B			50	E9	000F4	CALLS #3, DBG\$NGET_SYMID	
0A						BLBC R0, 8\$	

00000000G	00	08	AE	DD	000F7	PUSHL	SYMID LIST	7122
			01	FB	000FA	CALLS	#1, DBG\$STA_LOCK_SYMID	
			01	DD	00101	8\$: PUSHL	#1	7123
		10	AE	9F	00103	PUSHAB	DUMMY	
		28	A4	9F	00106	PUSHAB	40(R4)	
00000000G	00	10	AE	DD	00109	PUSHL	V VALDESC	
			04	FB	0010C	CALLS	#2, DBG\$NCPY_DESC	
			65	95	00113	9\$: TSTB	(R5)	7133
			05	12	00115	BNEQ	10\$	
		5F	A6	9F	00117	PUSHAB	P.ADT	7134
			12	11	0011A	BRB	12\$	
	01		65	91	0011C	10\$: CMPB	(R5), #1	7136
			05	12	0011F	BNEQ	11\$	
		6A	A6	9F	00121	PUSHAB	P.ADU	7137
			08	11	00124	BRB	12\$	
	02		65	91	00126	11\$: CMPB	(R5), #2	7139
			06	12	00129	BNEQ	13\$	
		75	A6	9F	0012B	PUSHAB	P.ADV	7140
	67		01	FB	0012E	12\$: CALLS	#1, DBG\$PRINT	
			01	A5	95	13\$: TSTB	1(R5)	7154
			03	13	00134	BEQL	14\$	
		0082	31	00136	BRW	27\$		
	02		65	91	00139	14\$: CMPB	(R5), #2	7161
			3B	13	0013C	BEQL	19\$	
		02	A5	95	0013E	TSTB	2(R5)	7167
			06	12	00141	BNEQ	15\$	
		0080	C6	9F	00143	PUSHAB	P.ADW	7168
			34	11	00147	BRB	20\$	
	01		02	A5	91	15\$: CMPB	2(R5), #1	7170
			06	12	0014D	BNEQ	16\$	
		008D	C6	9F	0014F	PUSHAB	P.ADX	7171
			28	11	00153	BRB	20\$	
	02		02	A5	91	16\$: CMPB	2(R5), #2	7173
			06	12	00159	BNEQ	17\$	
		009B	C6	9F	0015B	PUSHAB	P.ADY	7174
			1C	11	0015F	BRB	20\$	
	03		02	A5	91	17\$: CMPB	2(R5), #3	7176
			06	12	00165	BNEQ	18\$	
		00AA	C6	9F	00167	PUSHAB	P.ADZ	7177
			10	11	0016B	BRB	20\$	
	04		02	A5	91	18\$: CMPB	2(R5), #4	7179
			0D	12	00171	BNEQ	21\$	
		00AF	C6	9F	00173	PUSHAB	P.AEA	7180
			04	11	00177	BRB	20\$	
		00C0	C6	9F	00179	19\$: PUSHAB	P.AEB	7190
			01	FB	0017D	20\$: CALLS	#1, DBG\$PRINT	
0F		19	A4	05	E0	21\$: BBS	#5, 25(R4), 22\$	7198
7E		2C	A4	02	C3	SUBL3	#2, 44(R4), -(SP)	7199
	00000000G	00	01	FB	0018A	CALLS	#1, DBG\$IS_IT_ENTRY	
		07	50	E9	00191	BLBC	R0, 23\$	
			06	9F	00194	22\$: PUSHAB	P.AEC	7201
		67	01	FB	00198	CALLS	#1, DBG\$PRINT	
			28	A4	DD	23\$: PUSHL	40(R4)	7203
		6A	01	FB	0019E	CALLS	#1, DBG\$PRINT IDENTIFIER	
03		18	A4	04	E0	BBS	#4, 24(R4), 24\$	7209
			00BA	31	001A6	BRW	47\$	
06		18	A4	05	E1	24\$: BBC	#5, 24(R4), 25\$	7212

		00CE	C6	9F	001AE	PUSHAB	P.AED	: 7214
			04	11	001B2	BRB	26\$: 7217
		00D8	C6	9F	001B4	PUSHAB	P.AEE	: 7226
			00A5	31	001B8	BRW	46\$: 7232
	01	01	A5	91	001BB	CMPB	1(R5), #1	: 7235
			03	13	001BF	BECL	28\$: 7237
			0092	31	001C1	BRW	45\$: 7240
	52	18	A4	9E	001C4	MOVAB	24(R4), R2	: 7247
11	62		04	E1	001C8	BBC	#4, (R2), 31\$: 7250
06	62		05	E1	001CC	BBC	#5, (R2), 29\$: 7252
		00E4	C6	9F	001D0	PUSHAB	P.AEF	: 7255
			04	11	001D4	BRB	30\$: 7262
		00EC	C6	9F	001D6	PUSHAB	P.AEG	: 7265
	67		01	FB	001DA	CALLS	#1, DBG\$PRINT	: 7267
11	62		06	E1	001DD	BBC	#6, (R2), 34\$: 7270
			62	95	001E1	TSTB	(R2)	: 7281
			06	18	001E3	BGEQ	32\$: 7282
		00F6	C6	9F	001E5	PUSHAB	P.AEH	: 7283
			04	11	001E9	BRB	33\$: 7284
		0100	C6	9F	001EB	PUSHAB	P.AEI	: 7285
	67		01	FB	001EF	CALLS	#1, DBG\$PRINT	: 7286
11	62		02	E1	001F2	BBC	#2, (R2), 37\$: 7287
06	62		03	E1	001F6	BBC	#3, (R2), 35\$: 7288
		0108	C6	9F	001FA	PUSHAB	P.AEJ	: 7289
			04	11	001FE	BRB	36\$: 7290
		010E	C6	9F	00200	PUSHAB	P.AEK	: 7299
	67		01	FB	00204	CALLS	#1, DBG\$PRINT	: 7300
	05	02	A5	91	00207	CMPB	2(R5), #5	: 7307
			07	12	0020B	BNEQ	38\$: 7307
	50	011C	C6	9E	0020D	MOVAB	P.AEM, R0	: 7307
			37	11	00212	BRB	43\$: 7307
	06	02	A5	91	00214	CMPB	2(R5), #6	: 7307
			07	12	00218	BNEQ	39\$: 7307
	50	0123	C6	9E	0021A	MOVAB	P.AEN, R0	: 7307
			2A	11	0021F	BRB	43\$: 7307
	08	02	A5	91	00221	CMPB	2(R5), #8	: 7307
			07	12	00225	BNEQ	40\$: 7307
	50	012D	C6	9E	00227	MOVAB	P.AEO, R0	: 7307
			1D	11	0022C	BRB	43\$: 7307
	09	02	A5	91	0022E	CMPB	2(R5), #9	: 7307
			07	12	00232	BNEQ	41\$: 7307
	50	013A	C6	9E	00234	MOVAB	P.AEP, R0	: 7307
			10	11	00239	BRB	43\$: 7307
	07	02	A5	91	0023B	CMPB	2(R5), #7	: 7307
			05	13	0023F	BEQL	42\$: 7307
	7E		01	CE	00241	MNEGL	#1, -(SP)	: 7307
			07	11	00244	BRB	44\$: 7307
	50	014A	C6	9E	00246	MOVAB	P.AEQ, R0	: 7307
			50	DD	0024B	PUSHL	R0	: 7307
		0114	C6	9F	0024D	PUSHAB	P.AEL	: 7307
	67		02	FB	00251	CALLS	#2, DBG\$PRINT	: 7307
			0D	11	00254	BRB	47\$: 7307
	02	01	A5	91	00256	CMPB	1(R5), #2	: 7307
			07	12	0025A	BNEQ	47\$: 7307
		0150	C6	9F	0025C	PUSHAB	P.AER	: 7307
	67		01	FB	00260	CALLS	#1, DBG\$PRINT	: 7307
	07	03	A5	E9	00263	BLBC	3(R5), 48\$: 7307

		015E	C6	9F	00267	PUSHAB	P.AES		
			01	FB	00268	CALLS	#1, DBG\$PRINT		
07	67		1A	E1	0026E	BBC	#26, (R5), 49\$	7312	
	65	0168	C6	9F	00272	PUSHAB	P.AET		
	67		01	FB	00276	CALLS	#1, DBG\$PRINT		
	01	01	A5	91	00279	CMPB	1(R5), #1	7319	
			43	12	0027D	BNEQ	53\$		
		28	A4	D5	0027F	TSTL	40(R4)	7320	
			3E	13	00282	BEQL	53\$		
	55	28	A4	D0	00284	MOVL	40(R4), OPCODE_LIST	7335	
	52		01	CE	00288	MNEGL	#1, COLUMN	7336	
			53	D4	0028B	CLRL	OPCODE	7339	
29	65		53	E1	0028D	BBC	OPCODE, (OPCODE_LIST), 52\$		
	50	01	A2	9E	00291	MOVAB	1(R2), R0	7342	
52	50		00	EF	00295	EXTZV	#0, #3, R0, COLUMN		
	03		03	12	0029A	BNEQ	51\$	7343	
	68		00	FB	0029C	CALLS	#0, DBG\$NEWLINE		
	50	00000000G0043	0A	3C	0029F	MOVZWL	DBG\$OPCODE_KIND_TABLE[OPCODE], INDEX	7344	
	50		0A	C4	002A7	MULL2	#10, R0	7346	
		00000000G0040	06	9F	002AA	PUSHAB	DBG\$OPCODE_NAME_TABLE[R0]		
			06	DD	002B1	PUSHL	#6		
		0175	C6	9F	002B3	PUSHAB	P.AEU	7345	
	67		03	FB	002B7	CALLS	#3, DBG\$PRINT	7346	
CB	53	000001FF	8F	F3	002BA	AOBLEQ	#511, OPCODE, 50\$	7337	
	68		00	FB	002C2	CALLS	#0, DBG\$NEWLINE	7358	
	01	1C	A4	D1	002C5	CMPL	28(R4), #1	7363	
			0D	13	002C9	BEQL	54\$		
		1C	A4	DD	002CB	PUSHL	28(R4)	7366	
		017B	C6	9F	002CE	PUSHAB	P.AEV		
	67		02	FB	002D2	CALLS	#2, DBG\$PRINT		
	68		00	FB	002D5	CALLS	#0, DBG\$NEWLINE	7367	
		20	A4	D5	002D8	TSTL	32(R4)	7373	
			1A	13	002DB	BEQL	55\$		
	50	20	A4	D0	002DD	MOVL	32(R4), WHEN_ENTRY	7379	
	50	0C	A0	D0	002E1	MOVL	12(WHEN_ENTRY), STRING	7380	
		02	A0	9F	002E5	PUSHAB	2(STRING)	7381	
	7E		60	3C	002E8	MOVZWL	(STRING), -(SP)		
			6E	D7	002EB	DECL	(SP)		
		018A	C6	9F	002ED	PUSHAB	P.AEW		
	67		03	FB	002F1	CALLS	#3, DBG\$PRINT		
	68		00	FB	002F4	CALLS	#0, DBG\$NEWLINE	7382	
		24	A4	D5	002F7	TSTL	36(R4)	7388	
			1A	13	002FA	BEQL	56\$		
	50	24	A4	D0	002FC	MOVL	36(R4), DO_LIST_ENTRY	7394	
	50	0C	A0	D0	00300	MOVL	12(DO_LIST_ENTRY), STRING	7395	
		02	A0	9F	00304	PUSHAB	2(STRING)	7396	
	7E		60	3C	00307	MOVZWL	(STRING), -(SP)		
			6E	D7	0030A	DECL	(SP)		
		0198	C6	9F	0030C	PUSHAB	P.AEX		
	67		03	FB	00310	CALLS	#3, DBG\$PRINT		
	68		00	FB	00313	CALLS	#0, DBG\$NEWLINE	7397	
			04	00316	56\$:	RET		7400	

; Routine Size: 791 bytes, Routine Base: DBG\$CODE + 29E1


```
7294 7401 1 GLOBAL ROUTINE DELETE_EVENT_ENTRY(EVENT_ENTRY): NOVALUE =
7295 7402 1
7296 7403 1 FUNCTION
7297 7404 1 This routine is called to delete an event entry from its queue
7298 7405 1 (the command queue and event queue if linked in), as well as
7299 7406 1 any value descriptor, WHEN, or DO entries.
7300 7407 1
7301 7408 1 INPUTS
7302 7409 1 EVENT_ENTRY : Pointer to the event entry to be deleted.
7303 7410 1
7304 7411 1 OUTPUTS
7305 7412 1 None
7306 7413 1
7307 7414 1
7308 7415 2 BEGIN
7309 7416 2
7310 7417 2 MAP
7311 7418 2 EVENT_ENTRY : REF EVENT$EVENT_DESCRIPTOR;
7312 7419 2
7313 7420 2 LOCAL
7314 7421 2 WHEN_ENTRY : REF EVENT$WHEN_DESCRIPTOR, ! WHEN pointer
7315 7422 2 DO_LIST_ENTRY : REF EVENT$DO_LIST_DESCRIPTOR, ! DO List pointer
7316 7423 2
7317 7424 2 ADDRESS, ! Release address
7318 7425 2 SYMID_LIST,
7319 7426 2 DUMMY;
7320 7427 2
7321 7428 2
7322 7429 2 ! *****SSI
7323 7430 2 We just had Reserved Operand fault from RET of System service.
7324 7431 2 It is time to take the entry off, erase what we have done, just
7325 7432 2 as if nothing ever happened.
7326 7433 2
7327 7434 2 IF .EVENT_ENTRY [EVENT$B_CMD_TYPE] EQL EVENT$K_TYPE_SSINT
7328 7435 2 THEN
7329 7436 2 BEGIN
7330 7437 2 LOCAL
7331 7438 2 BIT_15_CNT: REF VECTOR[.BYTE],
7332 7439 2 BIT_15_CNT_SSV: REF VECTOR[.BYTE],
7333 7440 2 USERS_FP: REF BLOCK [ .BYTE];
7334 7441 2
7335 7442 2 BIT_15_CNT_SSV = .EVENT_ENTRY [EVENT$L_SSV_COUNT];
7336 7443 2 BIT_15_CNT = .EVENT_ENTRY [EVENT$L_FP_RET_CNT];
7337 7444 2
7338 7445 2
7339 7446 2 ! We have got reserved operand fault.
7340 7447 2
7341 7448 2 BIT_15_CNT[0] = .BIT_15_CNT[0] - 1;
7342 7449 2 USERS_FP = .EVENT_ENTRY [EVENT$L_USERS_FP];
7343 7450 2
7344 7451 2
7345 7452 2 ! This service is leaving.
7346 7453 2
7347 7454 2 IF .BIT_15_CNT[0] EQL 0
7348 7455 2 THEN
7349 7456 2 BEGIN
7350 7457 2
```

```
: 7351      7458  4
: 7352      7459  4      ! Get ready to report watched value at T-bit RET time.
: 7353      7460  4
: 7354      7461  4      IF .DBG$GB_SET_SSI_CNT THEN SKIP_WATCHES = TRUE;
: 7355      7462  4      (USERS_FP [SF$W_SAVE_PSW])<15,1,0> = FALSE;
: 7356      7463  4      ! Since we only have one entry on the event queue, so there is no need
: 7357      7464  4      to match the nested system calls.
: 7358      7465  4      BIT_15_CNT_SSV[0] = .BIT_15_CNT_SSV[0] - 1;
: 7359      7466  4
: 7360      7467  4      ! Just about to return to the caller.
: 7361      7468  4
: 7362      7469  4      BIT_15_CNT_SSV[0] = 0;
: 7363      7470  4
: 7364      7471  4      ! Cycle is all done.
: 7365      7472  4
: 7366      7473  4      DBG$GB_SET_WATCH_FLAG = FALSE;
: 7367      7474  4      END
: 7368      7475  4
: 7369      7476  4      ELSE
: 7370      7477  4
: 7371      7478  4
: 7372      7479  4      ! Still more left to go. More levels to go through, for example,
: 7373      7480  4      SDBG sees it, next IDBG should see it. Since we only make the
: 7374      7481  4      reserved operand fault goes through one level, this should never
: 7375      7482  4      happen.
: 7376      7483  4
: 7377      7484  4      (USERS_FP [SF$W_SAVE_PSW])<15,1,0> = TRUE;
: 7378      7485  4      END;
: 7379      7486  2
: 7380      7487  2
: 7381      7488  2      ! If this entry is a skips-event, remove it from its parent's queue
: 7382      7489  2      first. If its NOT a skips, and the parent queue is not empty, we
: 7383      7490  2      have (at the moment) and error... The parent still has active
: 7384      7491  2      offspring.
: 7385      7492  2
: 7386      7493  2      IF .EVENT_ENTRY [EVENT$B_CMD_TYPE] EQLU EVENT$K_TYPE_SKIPS OR
: 7387      7494  2      .EVENT_ENTRY [EVENT$B_CMD_TYPE] EQLU EVENT$K_TYPE_IGNORE OR
: 7388      7495  2      .EVENT_ENTRY [EVENT$B_CMD_TYPE] EQLU EVENT$K_TYPE_SSINT
: 7389      7496  2      THEN
: 7390      7497  2      REMQUE1 (.EVENT_ENTRY)
: 7391      7498  2
: 7392      7499  2      ELSE IF .EVENT_ENTRY [EVENT$L_EXC_FLINK] NEQA .EVENT_ENTRY OR
: 7393      7500  2      .EVENT_ENTRY [EVENT$L_EXC_BLINK] NEQA .EVENT_ENTRY
: 7394      7501  2      THEN
: 7395      7502  2      RETURN;
: 7396      7503  2
: 7397      7504  2
: 7398      7505  2      ! Remove the entry from the command queue.
: 7399      7506  2
: 7400      7507  2      REMQUE (.EVENT_ENTRY, EVENT_ENTRY);
: 7401      7508  2
: 7402      7509  2
: 7403      7510  2      ! If this is not a STEPS, SKIPS or IGNORE entry....
: 7404      7511  2
: 7405      7512  2      IF (.EVENT_ENTRY [EVENT$B_CMD_TYPE] NEQU EVENT$K_TYPE_STEPS) AND
: 7406      7513  2      (.EVENT_ENTRY [EVENT$B_CMD_TYPE] NEQU EVENT$K_TYPE_SKIPS) AND
: 7407      7514  2      (.EVENT_ENTRY [EVENT$B_CMD_TYPE] NEQU EVENT$K_TYPE_IGNORE) AND
```



```
: 7408 7515 3
: 7409 7516 3
: 7410 7517 3
: 7411 7518 3
: 7412 7519 3
: 7413 7520 3
: 7414 7521 3
: 7415 7522 3
: 7416 7523 3
: 7417 7524 3
: 7418 7525 4
: 7419 7526 4
: 7420 7527 4
: 7421 7528 4
: 7422 7529 4
: 7423 7530 4
: 7424 7531 4
: 7425 7532 5
: 7426 7533 5
: 7427 7534 5
: 7428 7535 5
: 7429 7536 5
: 7430 7537 5
: 7431 7538 5
: 7432 7539 5
: 7433 7540 5
: 7434 7541 5
: 7435 7542 5
: 7436 7543 5
: 7437 7544 6
: 7438 7545 6
: 7439 7546 6
: 7440 7547 6
: 7441 7548 6
: 7442 7549 6
: 7443 7550 6
: 7444 7551 7
: 7445 7552 7
: 7446 7553 7
: 7447 7554 7
: 7448 7555 7
: 7449 7556 7
: 7450 7557 7
: 7451 7558 7
: 7452 7559 7
: 7453 7560 7
: 7454 7561 8
: 7455 7562 8
: 7456 7563 8
: 7457 7564 8
: 7458 7565 8
: 7459 7566 8
: 7460 7567 8
: 7461 7568 8
: 7462 7569 8
: 7463 7570 8
: 7464 7571 9
```

```
(.EVENT_ENTRY [EVENT$B_CMD_TYPE] NEQU EVENT$K_TYPE_SSINT)
THEN
  BEGIN

    ! If the entry is an access type, release any primary
    ! and value descriptor.
    IF .EVENT_ENTRY [EVENT$B_CMD_KIND] EQLU EVENT$K_KIND_ACC
    THEN
      BEGIN

        ! Release any page entries.
        IF .EVENT_ENTRY [EVENT$B_SUB_KIND] EQLU EVENT$K_ACC_MDFY
        THEN
          BEGIN
            LOCAL
              PAGE_LIST : REF VECTOR [, LONG];

            ! Get the pages associated with this value
            IF DBGS$GET_PAGES (.EVENT_ENTRY [EVENT$L_PRIMARY],
                              PAGE_LIST,
                              DUMMY
                             )
            THEN
              BEGIN

                ! While there are entries in the page list, delete a
                ! page reference from the page queue.
                DO
                  BEGIN
                    LOCAL
                      PAGE_ENTRY : REF EVENT$PAGE_DESCRIPTOR;

                    ! Search through the existing entries in the page
                    ! queue for this page.
                    PAGE_ENTRY = .EVENT$PAGE_QUEUE [L_QUEUE_FLINK];
                    WHILE .PAGE_ENTRY NEQA EVENT$PAGE_QUEUE DO
                      BEGIN
                        IF .PAGE_ENTRY [EVENT$L_PAGE_ADDRESS] EQLA
                          .PAGE_LIST [1]
                        THEN

                          ! A match is found, so de-bump the
                          ! reference count, deleting the entry if
                          ! the count goes to zero.
                          BEGIN
```

```

: 7465 7572 9
: 7466 7573 9
: 7467 7574 9
: 7468 7575 9
: 7469 7576 10
: 7470 7577 10
: 7471 7578 10
: 7472 7579 10
: 7473 7580 9
: 7474 7581 9
: 7475 7582 8
: 7476 7583 8
: 7477 7584 8
: 7478 7585 8
: 7479 7586 8
: 7480 7587 8
: 7481 7588 7
: 7482 7589 7
: 7483 7590 7
: 7484 7591 7
: 7485 7592 6
: 7486 7593 5
: 7487 7594 4
: 7488 7595 4
: 7489 7596 4
: 7490 7597 4
: 7491 7598 4
: 7492 7599 4
: 7493 7600 4
: 7494 7601 4
: 7495 7602 4
: 7496 7603 4
: 7497 7604 4
: 7498 7605 4
: 7499 7606 4
: 7500 7607 4
: 7501 7608 4
: 7502 7609 4
: 7503 7610 4
: 7504 7611 4
: 7505 7612 4
: 7506 7613 4
: 7507 7614 4
: 7508 7615 4
: 7509 7616 5
: 7510 7617 5
: 7511 7618 5
: 7512 7619 5
: 7513 7620 5
: 7514 7621 5
: 7515 7622 5
: 7516 7623 5
: 7517 7624 5
: 7518 7625 5
: 7519 7626 5
: 7520 7627 5
: 7521 7628 5

```

```

PAGE_ENTRY [EVENT$W_PAGE_REF_COUNT] =
PAGE_ENTRY [EVENT$W_PAGE_REF_COUNT] - 1;
IF PAGE_ENTRY [EVENT$W_PAGE_REF_COUNT] EQLU 0
THEN
BEGIN
  REMQUE (.PAGE_ENTRY, PAGE_ENTRY);
  DBG$REL_MEMORY (.PAGE_ENTRY);
  EXITLOOP;
END;
END;

! Not found yet, point to the next one.
PAGE_ENTRY = .PAGE_ENTRY [EVENT$L_PAGE_FLINK];
END;

PAGE_LIST = .PAGE_LIST [0];
END
WHILE .PAGE_LIST;
END;
END;

! Notify the RST to release symid's.
IF DBG$NGET_SYMID (.EVENT_ENTRY [EVENT$L_PRIMARY],
  SYMID_LIST,
  DUMMY
)
THEN
  DBG$STA_UNLOCK_SYMID (.SYMID_LIST);

! Release the primary descriptor
DBG$NFREE_DESC (.EVENT_ENTRY [EVENT$L_PRIMARY], DUMMY);

! Release any value descriptor.
IF .EVENT_ENTRY [EVENT$V_HAS_VAL_DSCR]
THEN
  BEGIN

    ! Notify the RST to release symid's.
    IF DBG$NGET_SYMID (.EVENT_ENTRY [EVENT$L_VALDESC],
      SYMID_LIST,
      DUMMY
    )
    THEN
      DBG$STA_UNLOCK_SYMID (.SYMID_LIST);

```



```
: 7522      7629 5      ! Release the value descriptor
: 7523      7630 5      !
: 7524      7631 5      DBGS$FREE_DESC (.EVENT_ENTRY [EVENT$$_VALDESC], DUMMY);
: 7525      7632 4      END;
: 7526      7633 3      END;
: 7527      7634 3
: 7528      7635 3
: 7529      7636 3      ! Release any WHEN Entries.
: 7530      7637 3
: 7531      7638 3      WHEN ENTRY = .EVENT_ENTRY [EVENT$$_WHEN];
: 7532      7639 3      IF .WHEN_ENTRY NEQA 0
: 7533      7640 3      THEN
: 7534      7641 4          BEGIN
: 7535      7642 4
: 7536      7643 4
: 7537      7644 4          ! We have a WHEN entry. Decrement its reference count.
: 7538      7645 4          !
: 7539      7646 4          WHEN ENTRY [EVENT$$_WHEN_COUNT] =
: 7540      7647 4              .WHEN_ENTRY [EVENT$$_WHEN_COUNT] - 1;
: 7541      7648 4
: 7542      7649 4
: 7543      7650 4          ! Delete it if the count goes to 0 (or less !!!).
: 7544      7651 4          !
: 7545      7652 4          IF .WHEN_ENTRY [EVENT$$_WHEN_COUNT] LEQU 0
: 7546      7653 4          THEN
: 7547      7654 5              BEGIN
: 7548      7655 5
: 7549      7656 5
: 7550      7657 5          ! Release the expression buffer.
: 7551      7658 5          !
: 7552      7659 5          DBGS$REL_MEMORY (.WHEN_ENTRY [EVENT$$_WHEN_POINT]);
: 7553      7660 5
: 7554      7661 5
: 7555      7662 5          ! Release the WHEN descriptor.
: 7556      7663 5          !
: 7557      7664 5          DBGS$REL_MEMORY (.WHEN_ENTRY);
: 7558      7665 5          END
: 7559      7666 3      END;
: 7560      7667 3
: 7561      7668 3
: 7562      7669 3      ! Release any DO List Entries.
: 7563      7670 3      !
: 7564      7671 3      DO_LIST_ENTRY = .EVENT_ENTRY [EVENT$$_DO];
: 7565      7672 3      IF .DO_LIST_ENTRY NEQA 0
: 7566      7673 3      THEN
: 7567      7674 4          BEGIN
: 7568      7675 4
: 7569      7676 4
: 7570      7677 4          ! We have a DO List entry. Decrement its reference count.
: 7571      7678 4          !
: 7572      7679 4          DO_LIST_ENTRY [EVENT$$_DO_LIST_COUNT] =
: 7573      7680 4              .DO_LIST_ENTRY [EVENT$$_DO_LIST_COUNT] - 1;
: 7574      7681 4
: 7575      7682 4
: 7576      7683 4          ! Delete it if the count goes to 0 (or less !!!).
: 7577      7684 4          !
: 7578      7685 4          IF .DO_LIST_ENTRY [EVENT$$_DO_LIST_COUNT] LEQU 0
```



```
: 7579      7686      4
: 7580      7687      5
: 7581      7688      5
: 7582      7689      5
: 7583      7690      5
: 7584      7691      5
: 7585      7692      5
: 7586      7693      5
: 7587      7694      5
: 7588      7695      5
: 7589      7696      5
: 7590      7697      5
: 7591      7698      5
: 7592      7699      5
: 7593      7700      5
: 7594      7701      5
: 7595      7702      5
: 7596      7703      5
: 7597      7704      5
: 7598      7705      5
: 7599      7706      5
: 7600      7707      5
: 7601      7708      5
: 7602      7709      5
: 7603      7710      5
: 7604      7711      5
: 7605      7712      5
: 7606      7713      4
: 7607      7714      3
: 7608      7715      2
: 7609      7716      2
: 7610      7717      2
: 7611      7718      2
: 7612      7719      2
: 7613      7720      2
: 7614      7721      2
: 7615      7722      2
: 7616      7723      3
: 7617      7724      2
: 7618      7725      2
: 7619      7726      2
: 7620      7727      2
: 7621      7728      2
: 7622      7729      2
: 7623      7730      2
: 7624      7731      2
: 7625      7732      2
: 7626      7733      2
: 7627      7734      2
: 7628      7735      2
: 7629      7736      2
: 7630      7737      2
: 7631      7738      2
: 7632      7739      2
: 7633      7740      2
: 7634      7741      1

      THEN
      BEGIN
      LOCAL
        PREV_ENTRY: REF EVENT$DO_LIST_DESCRIPTOR,
        NEXT_ENTRY: REF EVENT$DO_LIST_DESCRIPTOR;

        ! Release the command buffer.
        !
        DBG$REL_MEMORY (.DO_LIST_ENTRY [EVENT$DO_LIST_POINT]);

        ! Unlink this DO_LIST entry from the list. The REMQUE
        ! instruction does the same as the following four lines
        ! (I hope).
        !
        PREV_ENTRY = .DO_LIST_ENTRY [EVENT$DO_LIST_BLINK];
        NEXT_ENTRY = .DO_LIST_ENTRY [EVENT$DO_LIST_FLINK];
        PREV_ENTRY [EVENT$DO_LIST_FLINK] = .NEXT_ENTRY;
        NEXT_ENTRY [EVENT$DO_LIST_BLINK] = .PREV_ENTRY;

        REMQUE (.DO_LIST_ENTRY, DO_LIST_ENTRY);

        ! Release the DO_LIST descriptor.
        !
        DBG$REL_MEMORY (.DO_LIST_ENTRY);
      END;
    END;
  END;

  ! Release the opcode list.
  !
  IF (.EVENT_ENTRY [EVENT$B_CMD_KIND] EQLU EVENT$K_KIND_INS) AND
    (.EVENT_ENTRY [EVENT$DO_OPCODE_LIST] NEQA 0) AND
    (.EVENT_ENTRY [EVENT$DO_OPCODE_LIST] NEQA DBG$OPCODES_CALL) AND
    (.EVENT_ENTRY [EVENT$DO_OPCODE_LIST] NEQA DBG$OPCODES_BRANCH)
  THEN
    DBG$REL_MEMORY (.EVENT_ENTRY [EVENT$DO_OPCODE_LIST]);

    ! Release Copied FP.
    !
    IF .EVENT_ENTRY [EVENT$B_CMD_TYPE] EQLU EVENT$K_TYPE_SKIPS OR
      .EVENT_ENTRY [EVENT$B_CMD_TYPE] EQLU EVENT$K_TYPE_STEPS
    THEN
      IF .EVENT_ENTRY [EVENT$DO_CALL_FRAME] NEQA 0
      THEN
        DBG$REL_MEMORY (.EVENT_ENTRY [EVENT$DO_CALL_FRAME]);

        ! Finally, release the Event Entry, itself.
        !
        DBG$REL_MEMORY (.EVENT_ENTRY);
      END;
```


				07FC 00000	.ENTRY	DELETE_EVENT_ENTRY, Save R2,R3,R4,R5,R6,R7,-;	
5A	00000000G	00	9E	00002	MOVAB	R8,R9,R10	7401
59	00000000G	00	9E	00009	MOVAB	DBG\$N\$FREE_DESC, R10	
58	00000000G	00	9E	00010	MOVAB	DBG\$STA_UNLOCK_SYMID, R9	
57	00000000'	EF	9E	00017	MOVAB	DBG\$N\$GET_SYMID, R8	
56	00000000'	EF	9E	0001E	MOVAB	EVENT\$PAGE_QUEUE, R7	
55	00000000G	00	9E	00025	MOVAB	SKIP_WATCHES, R6	
5E		0C	C2	0002C	MOVAB	DBG\$REL_MEMORY, R5	
52	04	AC	D0	0002F	SUBL2	#12, SP	
		54	D4	00033	MOVL	EVENT_ENTRY, R2	7434
06	14	A2	91	00035	CLRL	R4	
		2F	12	00039	CMPB	20(R2), #6	
		54	D6	0003B	BNEQ	3\$	
53	1C	A2	D0	0003D	INCL	R4	
51	2C	A2	D0	00041	MOVL	28(R2), BIT_15_CNT-SSV	7442
		61	97	00045	MOVL	44(R2), BIT_15_CNT-SSV	7443
50	24	A2	D0	00047	DECB	(BIT_15_CNT)	7448
		61	95	0004B	MOVL	36(R2), -USERS_FP	7449
		16	12	0004D	TSTB	(BIT_15_CNT)	7454
03	00000000G	00	E9	0004F	BNEQ	2\$	
66		01	D0	00056	BLBC	DBG\$GB_SET_SSI_CNT, 1\$	7461
05	A0	8F	8A	00059	MOVL	#1, SKIP_WATCHES	
		63	94	0005E	BICB2	#128, 5(USERS_FP)	7462
	CD	A7	94	00060	CLRB	(BIT_15_CNT-SSV)	7469
		05	11	00063	CLRB	DBG\$GB_SET_WATCH_FLAG	7473
05	A0	8F	88	00065	BRB	3\$	7454
04	14	A2	91	0006A	BISB2	#128, 5(USERS_FP)	7484
		09	13	0006E	CMPB	20(R2), #4	7493
	14	A2	91	00070	BEQL	4\$	
		03	13	00074	CMPB	20(R2), #5	7494
09		54	F9	00076	BEQL	4\$	
		52	DD	00079	BLBC	R4, 5\$	7495
FB86	CF	01	FB	0007B	PUSHL	R2	7497
		0D	11	00080	CALLS	#1, REMQUE1	
52	08	A2	D1	00082	BRB	7\$	
		04	12	00086	CMPL	8(R2), R2	7499
52	0C	A2	D1	00088	BNEQ	6\$	
		01	13	0008C	CMPL	12(R2), R2	7500
		04	0F	0008E	BEQL	7\$	
04	AC	62	0F	0008F	RET		
50	04	AC	D0	00093	REMQUE	(R2), EVENT_ENTRY	7507
03	14	A0	91	00097	MOVL	EVENT_ENTRY, R0	7512
		10	13	0009B	CMPB	20(R0), #3	
04	14	A0	91	0009D	BEQL	8\$	
		0A	13	000A1	CMPB	20(R0), #4	7513
05	14	A0	91	000A3	BEQL	8\$	
		04	13	000A7	CMPB	20(R0), #5	7514
06	14	A0	91	000A9	BEQL	8\$	
		03	12	000AD	CMPB	20(R0), #6	7515
		00C7	31	000AF	BNEQ	9\$	
	15	A0	95	000B2	BRW	20\$	
		03	13	000B5	TSTB	21(R0)	7523
					BEQL	10\$	

	02	16	008C	31	000B7	10\$:	BRW	18\$		
			A0	91	000BA		CMPB	22(R0), #2		7530
			41	12	000BE		BNEQ	15\$		
		08	AE	9F	000C0		PUSHAB	DUMMY		7539
		04	AE	9F	000C3		PUSHAB	PAGE_LIST		
		28	A0	DD	000C6		PUSHL	40(R0)		
00000000G	00		03	FB	000C9		CALLS	#3, DBG\$NGET_PAGES		
2E			50	E9	000D0		BLBC	R0, 15\$		
52			67	D0	000D3	11\$:	MOVL	EVENT\$PAGE_QUEUE, PAGE_ENTRY		7559
50			67	9E	000D6	12\$:	MOVAB	EVENT\$PAGE_QUEUE, R0		7560
50			52	D1	000D9		CMPL	PAGE_ENTRY, R0		
			1E	13	000DC		BEQL	14\$		
	50		6E	D0	000DE		MOVL	PAGE_LIST, R0		7563
04	A0	08	A2	D1	000E1		CMPL	8(PAGE_ENTRY), 4(R0)		
			0F	12	000E6		BNEQ	13\$		
		0C	A2	B7	000E8		DECW	12(PAGE_ENTRY)		7573
			0A	12	000EB		BNEQ	13\$		7574
	52		62	0F	000ED		REMQUE	(PAGE_ENTRY), PAGE_ENTRY		7577
			52	DD	000F0		PUSHL	PAGE_ENTRY		7578
	65		01	FB	000F2		CALLS	#1, DBG\$REL_MEMORY		
			05	11	000F5		BRB	14\$		7576
	52		62	D0	000F7	13\$:	MOVL	(PAGE_ENTRY), PAGE_ENTRY		7587
			DA	11	000FA		BRB	12\$		7560
			9E	DD	000FC	14\$:	PUSHL	@PAGE_LIST		7590
	D2		6E	E8	000FE		BLBS	PAGE_LIST, 11\$		7592
		08	AE	9F	00101	15\$:	PUSHAB	DUMMY		7599
		08	AE	9F	00104		PUSHAB	SYMID_LIST		
	52	04	AC	D0	00107		MOVL	EVENT_ENTRY, R2		
		28	A2	DD	0010B		PUSHL	40(R2)		
	68		03	FB	0010E		CALLS	#3, DBG\$NGET_SYMID		
06			50	E9	00111		BLBC	R0, 16\$		
		04	AE	DD	00114		PUSHL	SYMID_LIST		7604
	69		01	FB	00117		CALLS	#1, DBG\$STA_UNLOCK_SYMID		
		08	AE	9F	0011A	16\$:	PUSHAB	DUMMY		7609
		28	A2	DD	0011D		PUSHL	40(R2)		
	6A		02	FB	00120		CALLS	#2, DBG\$NFREE_DESC		
1E	17	A2	03	E1	00123		BBC	#3, 23(R2), 18\$		7614
		08	AE	9F	00128		PUSHAB	DUMMY		7621
		08	AE	9F	0012B		PUSHAB	SYMID_LIST		
		30	A2	DD	0012E		PUSHL	48(R2)		
	68		03	FB	00131		CALLS	#3, DBG\$NGET_SYMID		
06			50	E9	00134		BLBC	R0, 17\$		
		04	AE	DD	00137		PUSHL	SYMID_LIST		7626
	69		01	FB	0013A		CALLS	#1, DBG\$STA_UNLOCK_SYMID		
		08	AE	9F	0013D	17\$:	PUSHAB	DUMMY		7631
		30	A2	DD	00140		PUSHL	48(R2)		
	6A		02	FB	00143		CALLS	#2, DBG\$NFREE_DESC		
53		04	AC	D0	00146	18\$:	MOVL	EVENT_ENTRY, R3		7638
52		20	A3	D0	0014A		MOVL	32(R3), WHEN_ENTRY		
			10	13	0014E		BEQL	19\$		7639
		08	A2	D7	00150		DECL	8(WHEN_ENTRY)		7647
			0B	12	00153		BNEQ	19\$		7652
		0C	A2	DD	00155		PUSHL	12(WHEN_ENTRY)		7659
	65		01	FB	00158		CALLS	#1, DBG\$REL_MEMORY		
			52	DD	0015B		PUSHL	WHEN_ENTRY		7664
	65		01	FB	0015D		CALLS	#1, DBG\$REL_MEMORY		
52		24	A3	D0	00160	19\$:	MOVL	36(R3), DO_LIST_ENTRY		7671

		13	13	00164	BEQL	20\$		7672
	08	A2	D7	00166	DECL	8(DO_LIST_ENTRY)		7680
		0E	12	00169	BNEQ	20\$		7685
	0C	A2	DD	0016B	PUSHL	12(DO_LIST_ENTRY)		7695
65		01	FB	0016E	CALLS	#1, DBG\$REL_MEMORY		
52		62	0F	00171	REMQUE	(DO_LIST_ENTRY), DO_LIST_ENTRY		7707
		52	DD	00174	PUSHL	DO_LIST_ENTRY		7712
65		01	FB	00176	CALLS	#1, DBG\$REL_MEMORY		
52	04	AC	D0	00179	MOVL	EVENT_ENTRY, R2		7720
01	15	A2	91	0017D	CMPB	21(R2), #1		
		21	12	00181	BNEQ	21\$		
	28	A2	D5	00183	TSTL	40(R2)		7721
		1C	13	00186	BEQL	21\$		
50	0088	C6	9E	00188	MOVAB	DBG\$OPCODES_CALL, R0		7722
50	28	A2	D1	0018D	CMPL	40(R2), R0		
		11	13	00191	BEQL	21\$		
50	00C8	C6	9E	00193	MOVAB	DBG\$OPCODES_BRANCH, R0		7723
50	28	A2	D1	00198	CMPL	40(R2), R0		
		06	13	0019C	BEQL	21\$		
	28	A2	DD	0019E	PUSHL	40(R2)		7725
65		01	FB	001A1	CALLS	#1, DBG\$REL_MEMORY		
04	14	A2	91	001A4	CMPB	20(R2), #4		7730
		06	13	001A8	BEQL	22\$		
03	14	A2	91	001AA	CMPB	20(R2), #3		7731
		0B	12	001AE	BNEQ	23\$		
	3C	A2	D5	001B0	TSTL	60(R2)		7733
		06	13	001B3	BEQL	23\$		
	3C	A2	DD	001B5	PUSHL	60(R2)		7735
65		01	FB	001B8	CALLS	#1, DBG\$REL_MEMORY		
		52	DD	001BB	PUSHL	R2		7740
65		01	FB	001BD	CALLS	#1, DBG\$REL_MEMORY		
		04	001C0	RET				7741

; Routine Size: 449 bytes, Routine Base: DBG\$CODE + 2CF8

```
: 7636      7742 1 ROUTINE COMPARE_VMSDESC(DESC1, DESC2) =
: 7637      7743 1
: 7638      7744 1 FUNCTION
: 7639      7745 1     Determines whether two VMS descriptors represent the same area
: 7640      7746 1     of storage. This is used, for example, in processing the SET WATCH
: 7641      7747 1     command, to determine whether we already have a watchpoint on
: 7642      7748 1     the given data item.
: 7643      7749 1
: 7644      7750 1 INPUTS
: 7645      7751 1     DESC1 -           Points to a VMS descriptor
: 7646      7752 1     DESC2 -           Points to a VMS descriptor
: 7647      7753 1
: 7648      7754 1 OUTPUTS
: 7649      7755 1     The value TRUE or FALSE is returned.
: 7650      7756 1
: 7651      7757 2 BEGIN
: 7652      7758 2 MAP
: 7653      7759 2     DESC1: REF DBG$STG_DESC,
: 7654      7760 2     DESC2: REF DBG$STG_DESC;
: 7655      7761 2
: 7656      7762 2 LOCAL
: 7657      7763 2     LENGTH1,
: 7658      7764 2     LENGTH2,
: 7659      7765 2     OFFSET1,
: 7660      7766 2     OFFSET2;
: 7661      7767 2
: 7662      7768 2 ! Check whether the data given by the descriptors have
: 7663      7769 2 ! the same address.
: 7664      7770 2 !
: 7665      7771 2
: 7666      7772 2 IF .DESC1[DSC$B_CLASS] EQL DSC$K_CLASS_UBS
: 7667      7773 2 THEN OFFSET1 = .DESC1[DSC$L_POS]-ELSE OFFSET1 = 0;
: 7668      7774 2
: 7669      7775 2 IF .DESC2[DSC$B_CLASS] EQL DSC$K_CLASS_UBS
: 7670      7776 2 THEN OFFSET2 = .DESC2[DSC$L_POS]-ELSE OFFSET2 = 0;
: 7671      7777 2
: 7672      7778 2 IF (.DESC1[DSC$A_POINTER] + .OFFSET1<3,29,1>) NEQ
: 7673      7779 2 (.DESC2[DSC$A_POINTER] + .OFFSET2<3,29,1>) THEN RETURN FALSE;
: 7674      7780 2
: 7675      7781 2 IF .OFFSET1<0,3,0> NEQ .OFFSET2<0,3,0> THEN RETURN FALSE;
: 7676      7782 2
: 7677      7783 2 ! Compare the lengths.
: 7678      7784 2 ! (Do we really want to do this? The implications are:
: 7679      7785 2 !
: 7680      7786 2 ! SET WATCH ARRAY and SET WATCH ARRAY(1) are two different watchpoints,
: 7681      7787 2 ! since the watched areas have different lengths. This behavior seems
: 7682      7788 2 ! to be correct.
: 7683      7789 2 !
: 7684      7790 2 ! Say X is a byte variable at address 200. Say the default type is LONG.
: 7685      7791 2 ! Then SET WATCH X and SET WATCH 200 will be treated as two different
: 7686      7792 2 ! watchpoints (since one has length 1 and the other has length 4).
: 7687      7793 2 ! This does not seem to be a good thing. This problem could be solved
: 7688      7794 2 ! if we attempted to symbolize the "200" before setting the watchpoint
: 7689      7795 2 ! on it.)
: 7690      7796 2
: 7691      7797 2 LENGTH1 = DBG$DATA_LENGTH(.DESC1);
: 7692      7798 2 LENGTH2 = DBG$DATA_LENGTH(.DESC2);
```


: 7693
: 76947799 2
7800 1RETURN .LENGTH1 EQL .LENGTH2;
END;

```
007C 00000 COMPARE_VMSDESC:
56 00000000G 00 9E 00002 .WORD Save R2,R3,R4,R5,R6 : 7742
51 04 AC D0 00009 MOVAB DBG$DATA_LENGTH, R6 : 7772
0D 03 A1 91 0000D MOVL DESC1, R1
06 12 00011 CMPB 3(R1), #13
55 08 A1 D0 00013 BNEQ 1$
02 11 00017 MOVL 8(R1), OFFSET1 : 7773
55 D4 00019 1$: CLRL OFFSET1
52 08 AC D0 0001B 2$: MOVL DESC2, R2 : 7775
0D 03 A2 91 0001F CMPB 3(R2), #13
06 12 00023 BNEQ 3$
54 08 A2 D0 00025 MOVL 8(R2), OFFSET2 : 7776
02 11 00029 BRB 4$
54 D4 0002B 3$: CLRL OFFSET2
55 FD 8F 78 0002D 4$: ASHL #-3, OFFSET1, R3 : 7778
53 04 A1 C0 00032 ADDL2 4(R1), R3
54 FD 8F 78 00036 ASHL #-3, OFFSET2, R0 : 7779
50 04 A2 C0 0003B ADDL2 4(R2), R0
50 53 D1 0003F CMPL R3, R0
23 12 00042 BNEQ 6$
54 55 8D 00044 XORB3 OFFSET1, OFFSET2, R0 : 7781
07 50 93 00048 BITB R0, #7
1A 12 0004B BNEQ 6$
51 DD 0004D PUSHL R1 : 7797
66 01 FB 0004F CALLS #1, DBG$DATA_LENGTH
53 50 D0 00052 MOVL R0, LENGTH1 : 7798
52 DD 00055 PUSHL R2
66 01 FB 00057 CALLS #1, DBG$DATA_LENGTH : 7799
51 D4 0005A CLRL R1
50 53 D1 0005C CMPL LENGTH1, LENGTH2
02 12 0005F BNEQ 5$
51 D6 00061 INCL R1
50 51 D0 00063 5$: MOVL R1, R0
04 00066 RET
50 D4 00067 6$: CLRL R0
04 00069 RET : 7800
```

; Routine Size: 106 bytes, Routine Base: DBG\$CODE + 2EB9

: 7695 7801 1

```
: 7697 7802 1 ROUTINE GET_WATCH_POINT_VALUE (EVENT : REF EVENT$EVENT_DESCRIPTOR): NOVALUE =
: 7698 7803 1
: 7699 7804 1 FUNCTION
: 7700 7805 1
: 7701 7806 1 This routine gets the new value for a variable that is watched.
: 7702 7807 1 It does so by converting the Primary in the Event Entry to a
: 7703 7808 1 Value Descriptor which contains the value.
: 7704 7809 1
: 7705 7810 1 Along the way, this routine checks that the Primary is still
: 7706 7811 1 valid (i.e., still points to the same storage that it did when
: 7707 7812 1 the watchpoint was set). If it is not valid, the Primary is
: 7708 7813 1 replaced by a Volatile Value Descriptor, and then the watchpoint
: 7709 7814 1 is on an absolute address instead of a primary expression.
: 7710 7815 1
: 7711 7816 1 INPUTS
: 7712 7817 1 EVENT - Event entry pointer which describes the watchpoint.
: 7713 7818 1
: 7714 7819 1 OUTPUTS
: 7715 7820 1 The own variable NEWVALUE is set by this routine. The value in
: 7716 7821 1 NEWVALUE is then compared to the value in EVENT$EVENT$VALDESC
: 7717 7822 1 to see if the watched variable has changed (this comparison
: 7718 7823 1 is done in DBG$EXCEPTION_HANDLER, the called of this routine).
: 7719 7824 1
: 7720 7825 2 BEGIN
: 7721 7826 2 LOCAL
: 7722 7827 2 DUMMY, ! Output parameter for subroutines
: 7723 7828 2 PRIMARY, ! Temporary copy of the Primary
: 7724 7829 2 SYMID LIST, ! Symid list for LOCK SYMID
: 7725 7830 2 VALDESC: REF DBG$VALDESC, ! Value descriptor with value of Primary
: 7726 7831 2 V_VALDESC: REF DBG$VALDESC; ! Volatile Value Descriptor
: 7727 7832 2
: 7728 7833 2
: 7729 7834 2 ! Copy the Primary since PRIM_TO_VAL sometimes has the side effect
: 7730 7835 2 of modifying its argument. This copy is done into temporary
: 7731 7836 2 memory (fourth parameter is FALSE). Then call PRIM_TO_VAL
: 7732 7837 2 and get a Volatile Value Descriptor (this will have
: 7733 7838 2 the real address of the watched variable embedded in it).
: 7734 7839 2
: 7735 7840 2 DBG$NCOPY_DESC (.EVENT$EVENT$PRIMARY], PRIMARY, DUMMY, FALSE);
: 7736 7841 2 INVALID_FLAG = FALSE;
: 7737 7842 2 PRIM_TO_VAL (.PRIMARY, DBG$K_V_VALUE_DESC, V_VALDESC);
: 7738 7843 2
: 7739 7844 2
: 7740 7845 2 ! Check whether the Primary is still valid, i.e., whether it still
: 7741 7846 2 points to the same storage as it did when the watchpoint was set.
: 7742 7847 2
: 7743 7848 2 IF .INVALID_FLAG OR
: 7744 7849 2 CH$NEQ (T2, V_VALDESC[DBG$A_VALUE_VMSDESC], 12, EVENT$EVENT$A_VMSDESC))
: 7745 7850 2 THEN
: 7746 7851 2 BEGIN
: 7747 7852 2
: 7748 7853 2
: 7749 7854 2 ! Print an informational saying the Primary is no longer valid.
: 7750 7855 2
: 7751 7856 2 DBG$PRINT ($AC ('%DEBUG-I-WATCHVAR, watched variable '));
: 7752 7857 2 DBG$PRINT_IDENTIFIER(.EVENT$EVENT$PRIMARY]);
: 7753 7858 2 IF .INVALID_FLAG
```



```
: 7754      7859      3
: 7755      7860
: 7756      7861
: 7757      7862
: 7758      7863
: 7759      7864
: 7760      7865
: 7761      7866
: 7762      7867
: 7763      7868
: 7764      7869
: 7765      7870
: 7766      7871
: 7767      7872
: 7768      7873
: 7769      7874
: 7770      7875
: 7771      7876
: 7772      7877
: 7773      7878
: 7774      7879
: 7775      7880
: 7776      7881
: 7777      7882
: 7778      7883
: 7779      7884
: 7780      7885
: 7781      7886
: 7782      7887
: 7783      7888
: 7784      7889
: 7785      7890
: 7786      7891
: 7787      7892
: 7788      7893
: 7789      7894
: 7790      7895
: 7791      7896      1
```

```
THEN
  DBG$PRINT ($AC (' has gone out of scope'))
ELSE
  DBG$PRINT ($AC (' now points to a different address'));
DBG$NEWLINE();

! Free up the old Primary.
!
IF DBG$NGET_SYMID (.EVENT[EVENT$$_PRIMARY], SYMID_LIST, DUMMY)
THEN
  DBG$STA_UNLOCK_SYMID (.SYMID_LIST);
DBG$NFREE_DESC (.EVENT[EVENT$$_PRIMARY], DUMMY);

! Replace it with the Volatile Value Descriptor.
!
V_VALDESC = DBG$MAKE_VAL_DESC(EVENT[EVENT$$_VMSDESC], DBG$K_V_VALUE_DESC);
V_VALDESC[DBG$B_DHDR_LANG] = .DBG$GB_LANGUAGE;
IF DBG$NGET_SYMID (.V_VALDESC, SYMID_LIST, DUMMY)
THEN
  DBG$STA_LOCK_SYMID (.SYMID_LIST);
DBG$NCOPY_DESC (.V_VALDESC, EVENT[EVENT$$_PRIMARY], DUMMY, TRUE);
END;

! Build a Value Descriptor.
! Then copy the value descriptor into permanent memory.
! This is needed so that the storage stays around even if we
! execute DO clauses on the watchpoint (which may in turn
! cause calls to REL TEMPMEM).
! The memory for NEWVALUE is freed up in DBG$EXCEPTION_HANDLER
! when it is finished with the Value Descriptor.
DBG$PRIM TO VAL(.V_VALDESC, DBG$K_VALUE_DESC, VALDESC);
DBG$NCOPY_DESC (.VALDESC, NEWVALUE, DUMMY, TRUE);
RETURN 0;
END;
```

```
                                .PSECT  DBG$PLIT,NOWRT,  SHR,  PIC,0
48 43 54 41 57 2D 49 2D 47 55 42 45 44 25 24 0054B P.AEY: .ASCII  \$_DEBUG-I-WATCHVAR, watched variable \
61 76 20 64 65 68 63 74 61 77 20 2C 52 41 56 0055A
                                20 65 6C 62 61 69 72 00569
20 74 75 6F 20 65 6E 6F 67 20 73 61 68 20 16 00570 P.AEZ: .ASCII  <22>\ has gone out of scope\
                                65 70 6F 63 73 20 66 6F 0057F
6F 74 20 73 74 6E 69 6F 70 20 77 6F 6E 20 22 00587 P.AFA: .ASCII  \" now points to a different address\
64 61 20 74 6E 65 72 65 66 66 69 64 20 61 20 00596
                                73 73 65 72 64 005A5
```

```
                                .PSECT  DBG$CODE,NOWRT,  SHR,  PIC,0
03FC 00000 GET_WATCH_POINT_VALUE:
```

			59	00000000G	00	9E	00002	.WORD	Save R2,R3,R4,R5,R6,R7,R8,R9	7802		
			58	00000000G	00	9E	00009	MOVAB	DBG\$NGET_SYMID, R9			
			57	00000000G	00	9E	00010	MOVAB	DBG\$PRINT, R8			
			56	00000000G	EF	9E	00017	MOVAB	DBG\$NCOPY_DESC, R7			
			55	00000000G	EF	9E	0001E	MOVAB	P.AEY, R6			
			5E	00000000G	14	C2	00025	MOVAB	INVALID_FLAG, R5			
					7E	D4	00028	SUBL2	#20, SP	7840		
					14	AE	9F	0002A	CLRL	-(SP)		
					08	AE	9F	0002D	PUSHAB	DUMMY		
			54	04	AC	D0	00030	PUSHAB	PRIMARY			
					28	A4	DD	00034	MOVL	EVENT, R4		
			67	04	FB	00037		PUSHL	40(R4)			
					65	D4	0003A	CALLS	#4, DBG\$NCOPY_DESC	7841		
					04	AE	9F	0003C	CLRL	INVALID_FLAG	7842	
			7E	83	8F	9A	0003F	PUSHAB	V_VALDESC			
					08	AE	DD	00043	MOVZBL	#T31, -(SP)		
		0000V		CF	03	FB	00046	PUSHL	PRIMARY			
				OF	65	E8	0004B	CALLS	#3, PRIM TO VAL			
				50	04	AE	D0	0004E	BLBS	INVALID_FLAG, 1\$	7848	
34	A4			A0	0C	29	00052	MOVL	V_VALDESC, R0	7849		
					03	12	00058	CMPC3	#T2, 20(R0), 52(R4)			
					008D	31	0005A	BNEQ	1\$			
					56	DD	0005D	BRW	6\$		7856	
			68		01	FB	0005F	PUSHL	R6			
					28	A4	DD	00062	CALLS	#1, DBG\$PRINT	7857	
		00000000G	00		01	FB	00065	PUSHL	40(R4)			
			05		65	E9	0006C	CALLS	#1, DBG\$PRINT_IDENTIFIER			
					25	A6	9F	0006F	BLBC	INVALID_FLAG, 2\$	7858	
					03	11	00072	PUSHAB	P.AEZ	7860		
					3C	A6	9F	00074	BRB	3\$		
			68		01	FB	00077	PUSHAB	P.AFA	7862		
		00000000G	00		00	FB	0007A	CALLS	#1, DBG\$PRINT			
					10	AE	9F	00081	CALLS	#0, DBG\$NEWLINE	7863	
					0C	AE	9F	00084	PUSHAB	DUMMY	7868	
					28	A4	DD	00087	PUSHAB	SYMID_LIST		
			69		03	FB	0008A	PUSHL	40(R4)			
			0A		50	E9	0008D	CALLS	#3, DBG\$NGET_SYMID			
		00000000G	00		08	AE	DD	00090	BLBC	R0, 4\$	7870	
					01	FB	00093	PUSHL	SYMID_LIST			
					10	AE	9F	0009A	CALLS	#1, DBG\$STA_UNLOCK_SYMID		
					28	A4	DD	0009D	PUSHAB	DUMMY	7871	
		00000000G	00		02	FB	000A0	PUSHL	40(R4)			
			7E		83	8F	9A	000A7	CALLS	#2, DBG\$NFREE_DESC		
					34	A4	9F	000AB	MOVZBL	#131, -(SP)	7876	
		00000000G	00		02	FB	000AE	PUSHAB	52(R4)			
			04		50	D0	000B5	CALLS	#2, DBG\$MAKE_VAL_DESC			
04	BE			AE	00	F0	000B9	MOVL	R0, V_VALDESC			
		08		18	00000000G	00	9F	000C3	INSV	DBG\$GB_LANGUAGE, #24, #8, @V_VALDESC	7877	
					10	AE	9F	000C6	PUSHAB	DUMMY	7878	
					0C	AE	9F	000C9	PUSHAB	SYMID_LIST		
			69		03	FB	000CC	PUSHL	V_VALDESC			
			0A		50	E9	000CF	CALLS	#3, DBG\$NGET_SYMID			
		00000000G	00		08	AE	DD	000D2	BLBC	R0, 5\$	7880	
					01	FB	000D5	PUSHL	SYMID_LIST			
					01	DD	0C0DC	CALLS	#1, DBG\$STA_LOCK_SYMID			
					14	AE	9F	000DE	PUSHL	#1	7881	
								PUSHAB	DUMMY			

DBGEVENT
V04-000

L 6
16-Sep-1984 00:59:10
14-Sep-1984 12:16:53

VAX-11 Bliss-32 V4.0-742
[DEBUG.SRC]DBGEVENT.B32;1

Page 223
(29)

	28	A4	9F	000E1	PUSHAB	40(R4)	
	10	AE	DD	000E4	PUSHL	V_VALDESC	
67		04	FB	000E7	CALLS	#4, DBG\$NCOPY_DESC	
	0C	AE	9F	000EA	PUSHAB	VALDESC	7893
7E	7A	8F	9A	000ED	MOVZBL	#122, -(SP)	
	0C	AE	DD	000F1	PUSHL	V_VALDESC	
00000000G 00		03	FB	000F4	CALLS	#3, DBG\$PRIM_TO_VAL	
		01	DD	000FB	PUSHL	#1	7894
	14	AE	9F	000FD	PUSHAB	DUMMY	
	04	A5	9F	00100	PUSHAB	NEWVALUE	
	18	AE	DD	00103	PUSHL	VALDESC	
67		04	FB	00106	CALLS	#4, DBG\$NCOPY_DESC	
		04	00	00109	RET		7896

; Routine Size: 266 bytes, Routine Base: DBG\$CODE + 2F23

```

: 7793 7897 1 ROUTINE PRIM_TO_VAL (PRIM_DESC, TARGET_TYPE, VAL_DESC) : NOVALUE =
: 7794 7898 1
: 7795 7899 1 FUNCTION
: 7796 7900 1 This routine is a cover routine for DBG$PRIM_TO_VAL. It
: 7797 7901 1 establishes a handler to catch error signals from DBG$PRIM_TO_VAL,
: 7798 7902 1 and then calls DBG$PRIM_TO_VAL to do the work.
: 7799 7903 1
: 7800 7904 1 INPUTS
: 7801 7905 1 See DBG$PRIM_TO_VAL in DBGVALUES.
: 7802 7906 1
: 7803 7907 1 OUTPUTS
: 7804 7908 1 See DBG$PRIM_TO_VAL in DBGVALUES.
: 7805 7909 1
: 7806 7910 2 BEGIN
: 7807 7911 2 ENABLE
: 7808 7912 2 PRIM_TO_VAL_HANDLER;
: 7809 7913 2 DBG$PRIM_TO_VAL(.PRIM_DESC, .TARGET_TYPE, .VAL_DESC);
: 7810 7914 1 END;

```

```

0000 00000 PRIM_TO_VAL:
        6D      0010  CF  DE 00002      .WORD  Save nothing
        7E      08   AC  7D 00007      MOVAL  1$, (FP)
        04      04   AC  DD 0000B      MOVQ   TARGET_TYPE, -(SP)
00000000G 00      03   FB 0000E      PUSHL  PRIM_DESC
        04      00015      CALLS  #3, DBG$PRIM_TO_VAL
        0000 00016 1$:      RET
        7E      D4 00018      .WORD  Save nothing
        5E      DD 0001A      CLRL   -(SP)
        7E      04   AC  7D 0001C      PUSHL  SP
0000V  CF      03   FB 00020      MOVQ   4(AP), -(SP)
        04      00025      CALLS  #3, PRIM_TO_VAL_HANDLER
        04      00025      RET

```

; Routine Size: 38 bytes, Routine Base: DBG\$CODE + 302D

```

: 7897
: 7910
: 7913
:
: 7914
: 7910
:
:

```



```
7812 7915 1 ROUTINE PRIM_TO_VAL_HANDLER(SIGARG, MECHARG, ENBLARG) =
7813 7916 1
7814 7917 1 FUNCTION
7815 7918 1     This is the error handler for the routine PRIM_TO_VAL. If
7816 7919 1     an error occurs during the evaluation of a Primary then this
7817 7920 1     routine is invoked and it sets the own variable INVALID_FLAG
7818 7921 1     to indicate that the Primary is now invalid.
7819 7922 1
7820 7923 1 INPUTS
7821 7924 1     SIGARG - The signal argument vector.
7822 7925 1
7823 7926 1     MECHARG - The mechanism argument vector.
7824 7927 1
7825 7928 1     ENBLARG - The enable argument vector (EVENT pointer is passed in
7826 7929 1     from here).
7827 7930 1
7828 7931 1 OUTPUT
7829 7932 1     The OWN variable INVALID_FLAG is set to TRUE.
7830 7933 1
7831 7934 2 BEGIN
7832 7935 2 MAP
7833 7936 2     ENBLARG: REF VECTOR[.LONG],
7834 7937 2     SIGARG: REF VECTOR[.LONG];
7835 7938 2
7836 7939 2
7837 7940 2     ! If we were unwinding, continue unwinding.
7838 7941 2
7839 7942 2 IF .SIGARG[1] EQL SS$_UNWIND
7840 7943 2 THEN
7841 7944 2     RETURN SS$_CONTINUE;
7842 7945 2
7843 7946 2
7844 7947 2     ! Set the OWN flag saying the Primary is invalid.
7845 7948 2
7846 7949 2 INVALID_FLAG = TRUE;
7847 7950 2
7848 7951 2
7849 7952 2     ! Unwind to the caller of PRIM_TO_VAL.
7850 7953 2
7851 7954 2 SETUNWIND();
7852 7955 2 RETURN 0;
7853 7956 1 END;
```

```
0000 0000 PRIM_TO_VAL_HANDLER:
00000920 50 04 AC D0 00002 .WORD Save nothing
00000000' EF 04 A0 D1 00006 MOVL SIGARG, R0
00000000G 00 01 D0 00010 CMPL 4(R0), #2336
00000000G 00 01 D0 00013 BNEQ 1$
00000000G 00 01 D0 00014 RET #1, R0
00000000G 00 7E 7C 0001B MOVL #1, INVALID_FLAG
00000000G 00 02 FB 0001D CLRQ -(SP)
00000000G 00 02 FB 0001D CALLS #2, SYSSUNWIND
```

```
: 7915
: 7942
:
: 7944
: 7949
: 7954
:
```

DBGEVENT
V04-000

B 7
16-Sep-1984 00:59:10
14-Sep-1984 12:16:53

VAX-11 Bliss-32 V4.0-742
[DEBUG.SRC]DBGEVENT.B32;1

Page 226
(31)

50 D4 00024 CLRL R0
04 00026 RET

; 7955
; 7956

; Routine Size: 39 bytes, Routine Base: DBG\$CODE + 3053


```
7855 7957 1 ROUTINE PARSE_WHEN_CONDITION(EVENT): NOVALUE =
7856 7958 1
7857 7959 1 FUNCTION
7858 7960 1 This routine calls the parser to parse the when condition.
7859 7961 1 If any error occurs during the parsing, final handler catches
7860 7962 1 the error message, announce the message, WHEN_CONDITION is set
7861 7963 1 to FALSE in this case, Break point remains set.
7862 7964 1
7863 7965 1 INPUTS
7864 7966 1 EVENT - The address of the event_entry.
7865 7967 1
7866 7968 1 OUTPUTS
7867 7969 1 None.
7868 7970 1
7869 7971 1 BEGIN
7870 7972 2
7871 7973 2 MAP
7872 7974 2 EVENT: REF EVENT$EVENT_DESCRIPTOR; ! Event entry pointer
7873 7975 2
7874 7976 2 BUILTIN CALLG;
7875 7977 2
7876 7978 2 LOCAL
7877 7979 2
7878 7980 2 CONDITION, ! local when condition value
7879 7981 2 CONDITION_VAL : DBG$STG_DESC, ! Condition Value Descr
7880 7982 2 CONDITION_STR : DBG$STG_DESC, ! Condition String Descr
7881 7983 2 CONDITION_TXT : REF VECTOR [, WORD], ! Condition text
7882 7984 2 CONDITION_RDX, ! radix
7883 7985 2 MESSAGE_VECT, ! Message Vector
7884 7986 2 WHEN_ENTRY : REF EVENT$WHEN_DESCRIPTOR; ! WHEN Descriptor
7885 7987 2
7886 7988 2 ENABLE
7887 7989 2 DBG$FINAL_HANDL;
7888 7990 2
7889 7991 2
7890 7992 2 ! Assume WHEN condition is always true to begin with, so just in
7891 7993 2 ! case, bad things happened during the calls to other routine,
7892 7994 2 ! break point is still taken.
7893 7995 2
7894 7996 2 WHEN_CONDITION = TRUE;
7895 7997 2
7896 7998 2
7897 7999 2 ! Point to the WHEN entry.
7898 8000 2
7899 8001 2 WHEN_ENTRY = .EVENT [EVENT$L_WHEN];
7900 8002 2
7901 8003 2
7902 8004 2 ! Point to the WHEN expression.
7903 8005 2
7904 8006 2 CONDITION_TXT = .WHEN_ENTRY [EVENT$L_WHEN_POINT];
7905 8007 2
7906 8008 2
7907 8009 2 ! Set up the condition descriptors.
7908 8010 2
7909 8011 2 CONDITION_STR [DSC$B_CLASS] = DSC$K_CLASS_S;
7910 8012 2 CONDITION_STR [DSC$B_DTYPE] = DSC$K_DTYPE_T;
7911 8013 2 CONDITION_STR [DSC$W_LENGTH] = .CONDITION_TXT [0];
```



```

: 7912      8014 2      CONDITION_STR [DSC$A_POINTER] = CONDITION_TXT [1];
: 7913      8015 2
: 7914      8016 2      ! Set up some globals that C uses for purposes of preseving the
: 7915      8017 2      ! original casing.
: 7916      8018 2
: 7917      8019 2      IF .DBG$GB_LANGUAGE EQL DBG$K_C
: 7918      8020 2      THEN
: 7919      8021 2          BEGIN
: 7920      8022 2              DBG$GL_ORIG_COMMAND_PTR = CONDITION_TXT [1];
: 7921      8023 2              DBG$GL_UPCASE_COMMAND_PTR[0] = CONDITION_TXT [1];
: 7922      8024 2              DBG$GL_UPCASE_COMMAND_PTR[1] = CONDITION_TXT[1] + .CONDITION_TXT[0] - 1;
: 7923      8025 2              END;
: 7924      8026 2
: 7925      8027 2      ! Determine the current radix.
: 7926      8028 2
: 7927      8029 2      CONDITION_RDX = .DBG$GB_RADIX[DBG$B_RADIX_INPUT];
: 7928      8030 2
: 7929      8031 2
: 7930      8032 2      ! Obtain a value descriptor for the condition.
: 7931      8033 2
: 7932      8034 2      IF NOT DBG$NPARSE_EXPRESSION
: 7933      8035 2          (CONDITION_STR,
: 7934      8036 2           .CONDITION_RDX,
: 7935      8037 2           CONDITION,
: 7936      8038 2           TOKEN$K_TERM_NONE,
: 7937      8039 2           MESSAGE_VECT
: 7938      8040 2          )
: 7939      8041 2      THEN
: 7940      8042 2          CALLG (.MESSAGE_VECT, LIB$SIGNAL);
: 7941      8043 2
: 7942      8044 2
: 7943      8045 2      ! Set up the vax descriptor for the condition.
: 7944      8046 2      ! For now, we just declare the descriptor to be longword integer,
: 7945      8047 2      ! since this causes the fewest problems in the type converter.
: 7946      8048 2      ! Eventually, if we get a Boolean type and all languages support
: 7947      8049 2      ! it then we will build a target descriptor of this type.
: 7948      8050 2
: 7949      8051 2      CONDITION_VAL [DSC$B_CLASS] = DSC$K_CLASS_S;
: 7950      8052 2      CONDITION_VAL [DSC$B_DTYPE] = DSC$K_DTYPE_L;
: 7951      8053 2      CONDITION_VAL [DSC$W_LENGTH] = 4;
: 7952      8054 2      CONDITION_VAL [DSC$A_POINTER] = WHEN_CONDITION;
: 7953      8055 2      CONDITION_VAL [DSC$L_POS] = 0;
: 7954      8056 2
: 7955      8057 2
: 7956      8058 2      ! Special case for PASCAL. PASCAL returns descriptors
: 7957      8059 2      ! of type Boolean (dsc$K_dtype_tf) for relational expressions.
: 7958      8060 2
: 7959      8061 2      IF .DBG$GB_LANGUAGE EQL DBG$K_PASCAL
: 7960      8062 2      THEN
: 7961      8063 2          BEGIN
: 7962      8064 2              CONDITION_VAL [DSC$B_DTYPE] = DSC$K_DTYPE_TF;
: 7963      8065 2              CONDITION_VAL [DSC$W_LENGTH] = 1;
: 7964      8066 2              END;
: 7965      8067 2
: 7966      8068 2
: 7967      8069 2      ! Do the conversion from value descriptor to integer. NOTE THAT
: 7968      8070 2      ! I'M REUSING THE DESCRIPTOR, AND CONDITION LONGWORD.
```


7969 8071 2
7970 8072 2
7971 8073 2
7972 8074 2
7973 8075 2
7974 8076 2
7975 8077 2
7976 8078 2
7977 8079 2
7978 8080 2
7979 8081 2
7980 8082 1

```
!
IF NOT DBGSNTYPE_CONV (.CONDITION,
                        DBG$K_DEFAULT,
                        DBG$K_VAX_DESC,
                        CONDITION_VAL,
                        MESSAGE_VECT
                        )
THEN
    CALLG (.MESSAGE_VECT, LIB$SIGNAL);
RETURN 0;
END;
```

```
003C 00000 PARSE_WHEN CONDITION:
55 00000000' EF 9E 00002 .WORD Save R2,R3,R4,R5
54 00000000G 00 9E 00009 MOVAB WHEN CONDITION, R5
53 00000000G 00 9E 00010 MOVAB LIB$SIGNAL, R4
5E 00000000G 20 C2 00017 MOVAB DBG$GB_LANGUAGE, R3
6D 009D CF DE 0001A SUBL2 #32, SP
65 009D 01 D0 0001F MOVAL 5$, (FP)
50 04 AC D0 00022 MOVL #1, WHEN CONDITION
50 20 A0 D0 00026 MOVL EVENT, R0
50 0C A0 D0 0002A MOVL 32(R0), WHEN ENTRY
0A AE 010E 8F B0 0002E MOVL 12(WHEN ENTRY), CONDITION_TXT
52 60 3C 00034 MOVW #270, CONDITION_STR+2
08 AE 52 B0 00037 MOVW (CONDITION_TXT), R2
51 02 A0 9E 0003B MOVW R2, CONDITION_STR
0C AE 51 D0 0003F MOVAB 2(R0), R1
07 63 91 00043 MOVL R1, CONDITION_STR+4
00000000G 00 17 12 00046 CMPB DBG$GB_LANGUAGE, #7
00000000G 00 51 D0 00048 BNEQ 1$
00000000G 00 51 D0 0004F MOVL R1, DBG$GL_ORIG_COMMAND_PTR
00000000G 00 01 A240 9E 00056 MOVL R1, DBG$GL_UPCASE_COMMAND_PTR
MOVAB 1(R2)[CONDITION_TXT], -
DBG$GL_UPCASE_COMMAND_PTR+4
50 00000000G 00 9A 0005F 1$: MOVZBL DBG$GB_RADIX, CONDITION_RDX
04 AE 9F 00066 PUSHAB MESSAGE_VECT
7E D4 00069 CLRL -(SP)
08 AE 9F 0006B PUSHAB CONDITION
50 DD 0006E PUSHL CONDITION_RDX
18 AE 9F 00070 PUSHAB CONDITION_STR
00000000G 00 05 FB 00073 CALLS #5, DBG$NPARSE_EXPRESSION
04 50 E8 0007A BLBS R0, 2$
64 04 BE FA 0007D CALLG @MESSAGE_VECT, LIB$SIGNAL
14 AE 01080004 8F D0 00081 2$: MOVL #17301508, CONDITION_VAL
18 AE 65 9E 00089 MOVAB WHEN CONDITION, CONDITION_VAL+4
06 1C AE D4 0008D CLRL CONDITION_VAL+8
08 63 91 00090 CMPB DBG$GB_LANGUAGE, #6
16 AE 08 12 00093 BNEQ 3$
14 AE 28 90 00095 MOVW #40, CONDITION_VAL+2
04 01 B0 00099 MOVW #1, CONDITION_VAL
18 AE 9F 0009D 3$: PUSHAB MESSAGE_VECT
AE 9F 000A0 PUSHAB CONDITION_VAL
```

DBGEVENT
V04-000

F 7
16-Sep-1984 00:59:10
14-Sep-1984 12:16:53

VAX-11 Bliss-32 V4.0-742
[DEBUG.SRC]DBGEVENT.B32;1

Page 230
(32)

	7E	82	8F	9A	000A3	MOVZBL	#130, -(SP)
			01	DD	000A7	PUSHL	#1
		10	AE	DD	000A9	PUSHL	CONDITION
00000000G	00		05	FB	000AC	CALLS	#5, DBG\$NTYPE_CONV
	04		50	E8	000B3	BLBS	R0, 4\$
	64	04	BE	FA	000B6	CALLG	@MESSAGE_VECT, LIB\$SIGNAL
				04	000BA	RET	
				0000	000BB	4\$:	
						5\$:	
			7E	D4	000BD	.WORD	Save nothing
			5E	DD	000BF	CLRL	-(SP)
	7E	04	AC	7D	000C1	PUSHL	SP
00000000G	00		03	FB	000C5	MOVQ	4(AP), -(SP)
				04	000CC	CALLS	#3, DBG\$FINAL_HANDL
						RET	

8079
8082
7972

; Routine Size: 205 bytes, Routine Base: DBG\$CODE + 307A

; 7981 8083 1


```

: 7983      8084 1 ROUTINE MATCH_RIGHT_CALL_FRAME(EVENT) =
: 7984      8085 1
: 7985      8086 1 FUNCTION
: 7986      8087 1     This routine is used to match the exact call frame at the time
: 7987      8088 1     we set bit 15, or turn off bit 15 in saved PSW and the time we got
: 7988      8089 1     reserved operand fault.
: 7989      8090 1
: 7990      8091 1     For FP pointer may be changed inside of the call by the called
: 7991      8092 1     routine. So to be safe, we don't just compare the pointers, we
: 7992      8093 1     check some call frame contents as well.
: 7993      8094 1
: 7994      8095 1 INPUTS
: 7995      8096 1     Event entry for the event we consistier.
: 7996      8097 1
: 7997      8098 1 OUTPUTS
: 7998      8099 1     True for Yes, this is the frame caused the event,
: 7999      8100 1     Fasle for No, this is not the frame caused the event.
: 8000      8101 1
: 8001      8102 1
: 8002      8103 2 BEGIN
: 8003      8104 2 MAP
: 8004      8105 2     EVENT: REF EVENT$EVENT_DESCRIPTOR;
: 8005      8106 2
: 8006      8107 2 LOCAL
: 8007      8108 2     CALL_FRAME: REF BLOCK[.BYTE],    ! Pointer to FP
: 8008      8109 2     CALL_MASK: WORD,                ! Mask in Call frame
: 8009      8110 2     RUN_FRAME: REF BLOCK[.BYTE],    ! FP was saved in DEBUG run frame
: 8010      8111 2     RUN_MASK: WORD;                ! Mask in current FP
: 8011      8112 2
: 8012      8113 2
: 8013      8114 2 ! If there is no FP existed, simply returns.
: 8014      8115 2
: 8015      8116 2 IF .EVENT [EVENT$USER_FP] EQL 0 THEN RETURN FALSE;
: 8016      8117 2
: 8017      8118 2 ! Dig out current FP from DEBUG runframe.
: 8018      8119 2
: 8019      8120 2 RUN_FRAME = .DBG$RUNFRAME [DBG$USER_FP];
: 8020      8121 2
: 8021      8122 2 ! If the call frame saved in event entry is the same as current FP,
: 8022      8123 2 ! got it, no problem.
: 8023      8124 2
: 8024      8125 2 IF .EVENT [EVENT$USER_FP] EQL .RUN_FRAME
: 8025      8126 2 THEN
: 8026      8127 2     RETURN TRUE;
: 8027      8128 2
: 8028      8129 2 ! Need to do further investigation. Dig out the contents of the call
: 8029      8130 2 ! frame was saved from Call frame.
: 8030      8131 2
: 8031      8132 2 CALL_FRAME = .EVENT [EVENT$CALL_FRAME];
: 8032      8133 2 IF .CALL_FRAME EQL 0 THEN RETURN FALSE;
: 8033      8134 2
: 8034      8135 2 ! Compare the mask. But don't compare the stack alignment, compare
: 8035      8136 2 ! register mask and calls flag. Compare saved AP, and FP.
: 8036      8137 2
: 8037      8138 2 CALL_MASK = .CALL_FRAME[SFSW_SAVE_MASK];
: 8038      8139 2 RUN_MASK = .RUN_FRAME[SFSW_SAVE_MASK];
: 8039      8140 2 CALL_MASK = .CALL_MASK AND 'X'7FFF';
```

```
: 8040      8141  2  RUN_MASK = .RUN_MASK AND %X'7FFF';
: 8041      8142  2  IF (.CALL_MASK EQL .RUN_MASK) AND
: 8042      8143  2  (.CALL_FRAME[SF$$_SAVE_AP] EQL .RUN_FRAME[SF$$_SAVE_AP]) AND
: 8043      8144  2  (.CALL_FRAME[SF$$_SAVE_FP] EQL .RUN_FRAME[SF$$_SAVE_FP])
: 8044      8145  2  THEN
: 8045      8146  2  BEGIN
: 8046      8147  2
: 8047      8148  2      ! Got it, update the FP to current FP. So we can set/clear bit 15
: 8048      8149  2      ! in saved PSW in the correct FP.
: 8049      8150  2
: 8050      8151  2  EVENT [EVENT$$_USERS_FP] = .RUN_FRAME;
: 8051      8152  2  RETURN TRUE;
: 8052      8153  2  END
: 8053      8154  2  ELSE
: 8054      8155  2  RETURN FALSE;
: 8055      8156  2
: 8056      8157  1  END;
```

```
                                003C 00000 MATCH_RIGHT CALL FRAME:
                                .WORD Save R2,R3,R4,R5
                                50      04 AC D0 00002      MOVL EVENT, R0
                                24      A0 D5 00006      TSTL 36(R0)
                                46      13 00009      BEQL 2$
                                51 00000000G 00 D0 0000B      MOVL DBG$RUNFRAME+56, RUN_FRAME
                                51      24 A0 D1 00012      CMPL 36(R0), RUN_FRAME
                                35      13 00016      BEQL 1$
                                52      3C A0 D0 00018      MOVL 60(R0), CALL_FRAME
                                33      13 0001C      BEQL 2$
                                54      06 A2 B0 0001E      MOVW 6(CALL_FRAME), CALL_MASK
                                53      06 A1 B0 00022      MOVW 6(RUN_FRAME), RUN_MASK
                                55      0F 00 00026      EXTZV #0, #15, CALL_MASK, R5
                                55      0F 00 0002B      MOVW R5, CALL_MASK
                                55      0F 0002E      EXTZV #0, #15, RUN_MASK, R5
                                55      0F 00033      MOVW R5, RUN_MASK
                                54      B1 00036      CMPW CALL_MASK, RUN_MASK
                                16      12 00039      BNEQ 2$
                                08 A1      08 A2 D1 0003B      CMPL 8(CALL_FRAME), 8(RUN_FRAME)
                                0F      12 00040      BNEQ 2$
                                0C A1      0C A2 D1 00042      CMPL 12(CALL_FRAME), 12(RUN_FRAME)
                                08      12 00047      BNEQ 2$
                                24 A0      51 D0 00049      MOVL RUN_FRAME, 36(R0)
                                50      01 D0 0004D 1$:      MOVL #1, R0
                                04 00050      RET
                                50 D4 00051 2$:      CLRL R0
                                04 00053      RET
```

; Routine Size: 84 bytes, Routine Base: DBG\$CODE + 3147

```
: 8057      8158  1
: 8058      8159  1 END
: 8059      8160  1
: 8060      8161  0 ELUDOM

! End of module
```


.EXTRN LIB\$SIGNAL, SYSSUNWIND

PSECT SUMMARY

Name	Bytes	Attributes
DBG\$GLOBAL	69	NOVEC, WRT, RD, NOEXE, NOSHR, LCL, REL, CON, PIC, ALIGN(2)
DBG\$OWN	356	NOVEC, WRT, RD, NOEXE, NOSHR, LCL, REL, CON, PIC, ALIGN(2)
DBG\$CODE	12699	NOVEC, NOWRT, RD, EXE, SHR, LCL, REL, CON, PIC, ALIGN(0)
DBG\$PLIT	1450	NOVEC, NOWRT, RD, EXE, SHR, LCL, REL, CON, PIC, ALIGN(0)

Library Statistics

File	----- Total	Symbols Loaded	----- Percent	Pages Mapped	Processing Time
-\$255\$DUA28:[SYSLIB]LIB.L32;1	18619	42	0	1000	00:01.8
-\$255\$DUA28:[DEBUG.OBJ]STRUCDEF.L32;1	32	2	6	7	00:00.2
-\$255\$DUA28:[DEBUG.OBJ]DBGLIB.L32;1	1545	267	17	97	00:02.0
-\$255\$DUA28:[DEBUG.OBJ]DSTRECRDS.L32;1	418	106	25	31	00:00.4
-\$255\$DUA28:[DEBUG.OBJ]DBGMSG.L32;1	386	26	6	22	00:00.3
-\$255\$DUA28:[DEBUG.OBJ]DBGGEN.L32;1	150	7	4	12	00:00.3

COMMAND QUALIFIERS

; BLISS/CHECK=(FIELD,INITIAL,OPTIMIZE)/LIS=LIS\$:DBGEVENT/OBJ=OBJ\$:DBGEVENT MSRC\$:DBGEVENT/UPDATE=(ENH\$:DBGEVENT)

; Size: 12699 code + 1875 data bytes

; Run Time: 04:04.0

; Elapsed Time: 13:38.6

; Lines/CPU Min: 2006

; Lexemes/CPU-Min: 14391

; Memory Used: 1029 pages

; Compilation Complete

0082

**DIGITAL
CONFIDENTIAL**

EQUIPMENT
INITIAL AND

CORPORATION
PROPRIETARY

20
2Y

0083 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY